Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Peter Korcsok

Minimal Counterexamples to Flow Conjectures

Computer Science Institute of Charles University

Supervisor of the master thesis: Mgr. Robert Šámal, Ph.D.

Study programme: Computer Science

Specialization: Discrete Models and Algorithms

Prague 2015

Firstly, I would like to thank my supervisor, Mgr. Robert Šámal, Ph.D., for his valuable advices and ideas and for his time spent on our consultations.

Secondly, I want to thank all the Professors, Associate Professors and other teachers of Faculty of Mathematics and Physics of Charles University in Prague, especially those of Department of Applied Mathematics and Computer Science Institute of Charles University in Prague, for their lectures, where they showed me the beauty of mathematics and computer science.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

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

In Prague on May 5, 2015 Peter Korcsok

Název práce: Minimální protipříklady na hypotézy o tocích

Autor: Peter Korcsok

Ústav: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: Mgr. Robert Šámal, Ph.D.

Abstrakt: Říkáme, že graf má nikde-nulový k-tok, pokud umíme každé hraně přiradit její směr a přirozené číslo ($< k$) jako tok tak, aby pro každý vrchol v byl celkový přítok a odtok stejný. Tutte vyslovil v roce 1954 hypotézi, že každý graf bez mostů má nikde-nulový 5-tok, a tato hypotéza je stále otevřená. Kochol v nedávné práci představil výpočetní metodu na dokázání, že minimální protipříklad nemůže obsahovat krátkou kružnici (až do délky 10). V této práci poskytujeme ucelený přehled této metody a protože Kochol nezveřejnil svou implementaci (a pro nezávislé ověření metody), doplňujeme náš zdrojový kód, ktorý potvrzuje Kocholovy výsledky a rozšiřuje je: dokázali jsme, že minimální protipříklad neobsahuje kružnici kratší než 12.

Klíčová slova: nenulové toky, minimální protipříklad

Title: Minimal Counterexamples to Flow Conjectures

Author: Peter Korcsok

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Robert Šámal, Ph.D.

Abstract: We say that a graph admits a nowhere-zero k-flow if we can assign a direction and a positive integer ($< k$) as a flow to each edge so that total in-flow into v and total out-flow from v are equal for each vertex v. In 1954, Tutte conjectured that every bridgeless graph admits a nowhere-zero 5-flow and the conjecture is still open. Kochol in his recent papers introduces a computational method how to prove that a minimal counterexample cannot contain short circuits (up to length 10). In this Thesis, we provide a comprehensive view on this method. Moreover, since Kochol does not share his implementation and in order to independently verify the method, we provide our source code that validates Kochol's results and extend them: we prove that any minimal counterexample to the conjecture does not contain any circuit of length less than 12.

Keywords: nowhere zero flows, minimal counterexample

Contents

Introduction **1**

1 The Basics **2**
1.1 Flows and Colorings . . . 2
1.2 Historical Notes . . . 4

2 Restrictions on Counterexamples to the 5-Flow Conjecture **6**
2.1 Kochol's Approach . . . 7
2.1.1 Forbidden Networks . . . 9
2.1.2 The Computations . . . 12
2.2 Modifications of the Approach . . . 16

Conclusion **19**

Bibliography **20**

Lists of Figures, Tables and Code Snippets **22**
List of Figures . . . 22
List of Tables . . . 22
List of Code Snippets . . . 22

List of Notation **23**

Appendices **24**

A Source Code of the Programs **25**
A.1 Kochol's Basic Method . . . 25
A.2 Kochol's Advanced Method . . . 28
A.3 Modificated Advanced Method . . . 33

Introduction

The concept of a flow on a graph was introduced by Tutte [12] who noticed the connection between the flows and the colorings of graphs. Since 1954 when he stated his 5-Flow Conjecture, it is still an open problem. There were several attempts to prove the conjecture and many of them were formulated as studying some aspects of a hypothetical minimal counterexample.

Recently, Kochol [7, 8] has introduced a method using so-called forbidden networks, i.e. graphs that cannot be a subgraph of any such counterexample. He has proved that any minimal counterexample does not contain a circuit shorter than 11.

The aim of this thesis is to provide systematic and comprehensive view on Kochol's method. Moreover, since some part of the method requires computers and Kochol has not shared his implementation, we have created a program that validates Kochol's results and we have also improved the now-known best result using this program, see Theorem 2.10.

The Structure of the Thesis

In Chapter 1, we provide some introduction to the definitions needed to understand the problem of a nowhere-zero 5-flow. In the end of the chapter, we also provide some historical notes to illustrate how the knowledge of flows changed through the time.

The most important part of the thesis is Chapter 2. The reader can find there the motivational proof that short circuits (of length 3 and 4) cannot be subgraphs of any minimal counterexample to the 5-Flow Conjecture. Later, in Section 2.1, we provide the description and the proofs of the most important or interesting parts of Kochol's method.

In Subsection 2.1.2, some tricks how to reduce the size of computations are showed. And finally, in Section 2.2, we discuss possible modifications of this method and some results we obtained so far.

Source code used to validate Kochol's results can be found in Appendix A or on author's website `http://kam.mff.cuni.cz/~korcsok/masterthesis/`.

Chapter 1

The Basics

In graph theory, a *graph* G is considered as a pair of sets – a set of *vertices* denoted by V_G and a multiset of *edges* denoted by E_G where each edge $e \in E_G$ is a set of one or two vertices – usually called *ends* of the edge e. An edge where both ends are the same vertex is called a *loop*. In the following text, we will assume that all graphs are non-empty, i.e. $V_G \neq \emptyset$, and finite, i.e. $|V_G| < \infty$.

A graph H is a *subgraph* of G if $V_H \subseteq V_G$ and $E_H \subseteq E_G$. Given vertex set $W \subseteq V_G$, we denote by $G[W]$ the subgraph H of G where $V_H = W$ and edge $e \in E_G$ is element of E_H if and only if both its ends are elements of W. Subgraph $G[W]$ is also called the subgraph *induced* by vertices W.

Given a graph G, a *walk* of length k in G connecting vertices v_0 and v_k is a sequence $v_0, e_0, v_1, \ldots, e_{k-1}, v_k$ where $v_0, v_1, \ldots, v_k \in V_G$, $e_0, \ldots, e_{k-1} \in E_G$ and for each $i = 0, \ldots, k-1$ edge e_i has ends v_i and v_{i+1}. A walk where all vertices are distinct from each other is called a *path* and denoted P_k. Furthermore, a walk with all vertices pairwise distinct except that $v_0 = v_k$ is called a *circuit* and denoted by C_k.

We say that two vertices $u, v \in V_G$ are *connected* if there exists a walk in G connecting u and v. Clearly, the relation of "being connected" is an equivalence. Therefore, we can partition all vertices into disjoint subsets $V_1, \ldots, V_k$ such that two vertices are connected if and only if they are elements of the same set V_i.

The subgraphs $G[V_i]$ are called the *components* of graph G. Again, it is not difficult to see that there is no edge of G having its ends in distinct components. We a call a graph *connected* if it contains exactly one component.

Let $G - e$ be the graph obtained from G by deleting edge e. A *bridge* in a graph G is an edge $e \in E_G$ such that $G - e$ contains more components than G. Clearly, an edge e is a bridge in G if and only if there exist such two vertices $u, v \in V_G$ that every path in G connecting u and v contains the edge e. Alternatively, a bridge can be defined as the edge that is not contained in any circuit.

We call a graph *k-connected* if we can delete any $k-1$ edges without obtaining a graph with at least two components. Furthermore, we call a graph *cyclically k-connected* if deleting any $k - 1$ edges cannot create a graph with at least two components containing a circuit.

1.1 Flows and Colorings

The graph as was defined in the previous section is sometimes called *undirected* because each edge connects its ends in both ways. A *digraph* (or *directed* graph) D is defined similarly to a graph with set of *arcs* instead of edges. An arc $\vec{e} \in E_D$ is a pair of two (not necessary distinct) vertices where the first one is called a *tail* and the second one a *head* of the arc. An arc where both tail and head are the same vertex is again called a loop.

By an *orientation* $\vec{G}$ of an undirected graph G we understand a digraph where we assign a direction to each edge, i.e. for an edge we choose one end as a tail and the second as a head. Let $\vec{G}$ be some orientation of a graph G. For a vertex v we denote by $E^+_{\vec{G}}(v)$ (or $E^-_{\vec{G}}(v)$) the set of all arcs in $E_{\vec{G}}$ with tails (or heads, respectively) in the vertex v. If it is clear from context, we can omit the subscript.

Let Γ be an Abelian group and $f\colon E_{\vec{G}} \to \Gamma$ for some orientation $\vec{G}$ of a graph G. We define functions $f^+, f^-\colon V_G \to \Gamma$ as following:

$$f^+(v) = \sum_{\vec{e} \in E^+(v)} f(\vec{e}), \qquad\qquad f^-(v) = \sum_{\vec{e} \in E^-(v)} f(\vec{e}).$$

Definition 1.1. A Γ-*flow* on a graph G where Γ is an Abelian group is a mapping $f\colon E_{\vec{G}} \to \Gamma$ for some orientation $\vec{G}$ of graph G such that $f^+(v) = f^-(v)$ holds for every vertex v. A Γ-flow where $\Gamma = \mathbb{Z}_k$ is also called a k-*flow*. Moreover, a flow f is called *nowhere-zero* if $f(\vec{e}) \neq 0$ holds for each arc $\vec{e}$.

Observation 1.2. *Let $\vec{G}$ and $\vec{G}'$ be two different orientations of a graph G and let Γ be an Abelian group. Then there exists a flow $f\colon E_{\vec{G}} \to \Gamma$ if and only if there exists a flow $f'\colon E_{\vec{G}} \to \Gamma$. Moreover, flow f is nowhere-zero if and only if flow f' is.*

As a corollary, we can choose and fix an orientation $\vec{G}$ of G with no change of existence of a flow on G. We will use this fact later in the proof of Theorem 2.1 and in Section 2.1.

By a k-*coloring* of a graph G we understand a mapping $c\colon V_G \to \{1, \dots, k\}$ such that $c(u) \neq c(v)$ holds whenever vertices u and v are ends of the same edge.

The connection between nowhere-zero flows and colorings can be observed considering the duality of planar graphs. Therefore, we need to introduce two additional definitions.

We call a graph G *planar* if it can be drawn on a plane without crossing edges. More specifically, each vertex v is drawn as a point p_v and each edge e is drawn as a plane curve c_e such that points representing different vertices are distinct, the ends of c_e for $e = \{u, v\}$ are p_u and p_v and any two curves can intersect each other only in points representing their common vertices. The graph together with such drawing is called *plane* graph. The curves representing the edges of a plane graph G divide the plane into some areas, which we call *faces* of G.

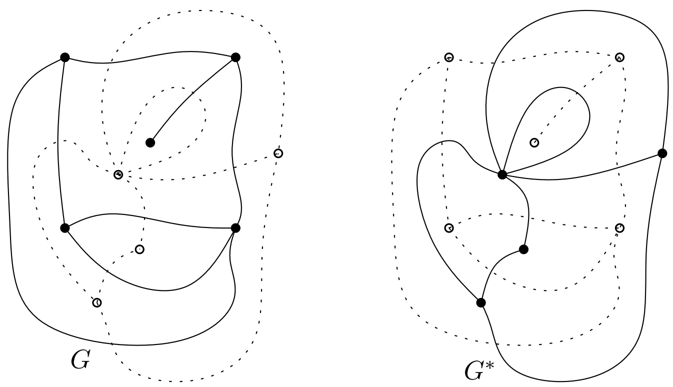

Figure 1.1: An example of a pair of dual graphs.

Given a plane graph G, a *dual* graph G^* is a graph that contains a vertex v_F^* for each face F of G and an edge e_e^* for each edge $e \in E_G$ connecting the vertices corresponding the faces of G lying on both sides of the edge e, see Figure 1.1. Clearly, e_e^* is a loop if and only if e is a bridge and vice versa.

Observation 1.3. *A planar graph G admits a nowhere-zero k-flow if and only if its dual graph G^* has a k-coloring.*

Proof. Let G' be some plane drawing of a graph G and G^* be its dual graph. By the definition of a dual graph, a k-coloring $c\colon V_{G^*} \to \{1, \ldots, k\}$ naturally corresponds to a coloring of faces of G' using colors $1, \ldots, k$ where no adjacent faces have the same color.

Using this face-coloring, we define an orientation $\vec{G}$ of G where each edge is oriented such that the face with larger color number lies on its right side. Furthermore, we define a mapping $f\colon E_{\vec{G}} \to \{1, \ldots, k-1\}$ as a (positive) difference between color numbers of the faces on both sides of the edge, see Figure 1.2.

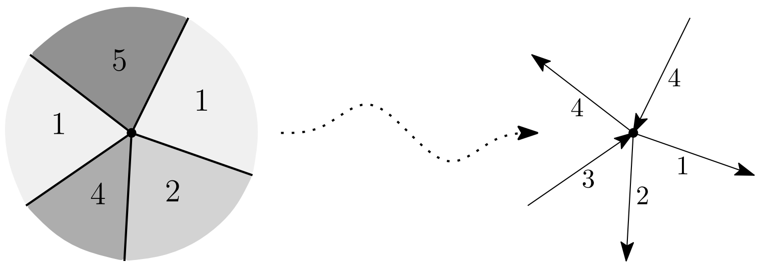

Figure 1.2: The definition of an orientation $\vec{G}$ and a mapping f according to face-coloring.

We show that f is a nowhere-zero k-flow on G. Clearly, it is sufficient to show that f is a flow. Let v be a vertex of G'. When we look at the colors of faces in a clockwise direction once around the vertex v starting and ending on the same face we get a sequence of numbers $1, \ldots, k$ where the first and the last numbers are the same. By the definition, $f^+(v)$ equals the sum of all increases in this sequence whereas $f^-(v)$ is the sum of all decreases. Therefore, $f^+(v) = f^-(v)$ must hold for any vertex v.

For the other implication, it suffices to find an arc with a maximal flow and color the face on its left side by color 1. All the other faces get the color in process inverse to the one displayed on Figure 1.2.

Similarly to the fist part of this proof, we can show that this process assigns to each face exactly one color and no two adjacent faces share the same one. ☐

1.2 Historical Notes

In 1939, Robbins [9] studied *strong* orientations of graphs, i.e. such orientations that there exists a path from each vertex to each other respecting the directions of arcs. He also proved that a graph can be strongly oriented if and only if it does not contain any bridge. Using a little modification of his proof, the following theorem can be proved.

Theorem 1.4. *For every bridgeless graph G there exists a natural number k such that G admits a nowhere-zero k-flow.*

In 1952, Tutte [12] studied some polynomials counting the numbers of various colorings of graphs and formulated the following conjectures.

Conjecture 1.5. *There exists a natural number k such that each bridgeless graph admits a nowhere-zero k-flow.*

Conjecture 1.6 (5-Flow Conjecture)**.** *Each bridgeless graph admits a nowhere-zero 5-flow.*

In 1976, Appel and Haken [1–3] proved the famous 4-Color Theorem saying that every planar graph with no loop has a 4-coloring. Therefore and from Observation 1.3, every bridgeless planar graph admits a nowhere-zero 4-flow.

Conjecture 1.5 was proved in 1976 by Jaeger and later, in 1980, improved by Seymour.

Theorem 1.7 (Jaeger [5])**.** *Each bridgeless graph admits a nowhere-zero 8-flow.*

Theorem 1.8 (Seymour [10])**.** *Each bridgeless graph admits a nowhere-zero 6-flow.*

Chapter 2

Restrictions on Counterexamples to the 5-Flow Conjecture

One possible approach to prove the 5-Flow Conjecture is to look at a hypothetical counterexample and prove some facts that must hold, e.g. Seymour [10] proved that a minimal counterexample to the 5-Flow Conjecture is *cubic*, i.e. each vertex has degree exactly 3, and 3-connected. Celmins in his Ph.D. thesis [4] proved that such minimal counterexample is cyclically 5-connected and does not contain any circuit of length less then 7.

As a simple introduction to this technique, we present the following theorem.

Theorem 2.1. *Neither C_3 nor C_4 can be a subgraph of a minimal counterexample to the 5-Flow Conjecture.*

Proof. For a contradiction, let G be some minimal counterexample to the 5-Flow Conjecture containing C_3 as a subgraph and denote the vertices of this subgraph by v_1, v_2 and v_3. As mentioned, G is cubic.

Let G' be the graph obtained from G by contracting all three edges of C_3, i.e. by removing edges $\{v_1, v_2\}$, $\{v_2, v_3\}$ and $\{v_3, v_1\}$ and identifying all three vertices into new vertex v, see Figure 2.1.

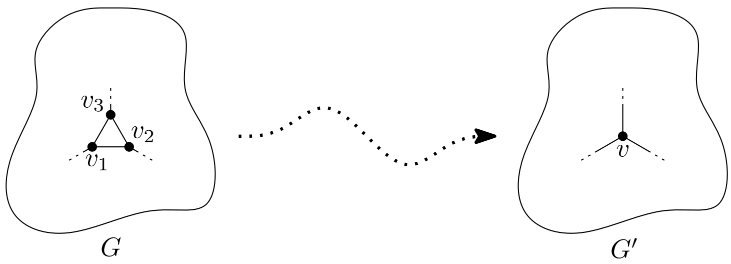

Figure 2.1: A contraction of C_3.

Clearly, G' is a smaller bridgeless graph, therefore, there exists a nowhere-zero 5-flow f' on G'. According to Observation 1.2, we can assume f' uses an orientation $\vec{G}'$ where no arc has its tail in v. Therefore, $E^+_{\vec{G}'}(v) = \emptyset$ and $f'^-(v) = a + b + c = 0$.

Let $\vec{G}$ be an orientation of G where each edge e satisfying $|e \cap \{v_1, v_2, v_3\}| \le 1$ has the same direction in $\vec{G}$ as in $\vec{G}'$ and edges $\{v_1, v_2\}$, $\{v_2, v_3\}$ and $\{v_3, v_1\}$ are oriented into arcs (v_1, v_2), (v_2, v_3) and (v_3, v_1), respectively, see Figure 2.2.

We can expand the flow f' into a 5-flow f on graph G such that $f(\vec{e}) = f'(\vec{e})$ for each edge $e \in E_{G'}$. Let $f((v_1, v_2)) = x$, then $f((v_2, v_3)) = x + a$ and $f((v_3, v_1)) = x + a + b$ where a and b are values of the flow f' on two arcs with head in v, see Figure 2.2. The flow f' is nowhere-zero and, therefore, values x,

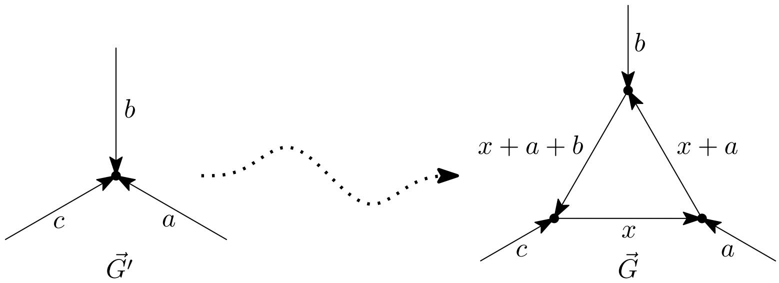

Figure 2.2: Flow in the neighborhood of C_3 before and after decontraction.

$x+a$ and $x+a+b$ are pairwise different. Therefore, there are two possible choices of x in $\mathbb{Z}_5$ such that all x, $x+a$ and $x+a+b$ are non-zero.

Both these choices lead to a nowhere-zero 5-flow on G and, therefore, G is not a counterexample to the 5-Flow Conjecture.

Similarly, we can expand some nowhere-zero 5-flow on G whenever it contains C_4 as a subgraph. $\square$

Kochol [6–8] has also chosen this approach and he has shown that any circuit in any minimal counterexample to 5-FC has length at least eleven. The purpose of Section 2.1 is to provide a simple overview of Kochol's methods and the proofs of their correctness.

In Section 2.2, we provide some modifications of this approach we have studied and results we obtained.

2.1 Kochol's Approach

In 2006, Kochol [7] has introduced a computational method to prove that a certain graph cannot be a subgraph of a minimal counterexample to the 5-Flow Conjecture.

A *network* is a pair (G, U) where G is a graph and $U = \{u_1, \ldots, u_n\}$ is an ordered set of pairwise distinct vertices of G, the vertices U are also called *terminals* of network (G, U). Without loss of generality, we can assume that no two terminals are connected by an edge; in the other case we can subdivide the edge with a new non-terminal vertex. Later, we will split a graph into two networks, which will make it easier to analyze existence of flows.

Similarly to the definition of a nowhere-zero 5-flow on a graph, a *nowhere-zero 5-flow* (or shortly a *flow*) on a network (G, U) is a mapping $f \colon E_{\vec{G}} \to \mathbb{Z}_5 \setminus \{0\}$ for some orientation $\vec{G}$ of G where $f^+(v) = f^-(v)$ holds for each vertex except of the terminals. Clearly, a mapping f is a flow on $(G, \emptyset)$ if and only if it is a nowhere-zero 5-flow on G.

A network (G, U) where all terminals $U = \{u_1, \ldots, u_n\}$ have degree 1 is called *simple*. We fix an orientation $\vec{G}$ of G such that there is no arc with a head in any terminal. We can do this without loss of generality according to Observation 1.2. Let f be a flow on (G, U), then we denote by $f^+(U)$ the n-tuple $(f^+(u_1), \ldots, f^+(u_n))$.

Lemma 2.2. *Let (G,U) be a simple network and f a flow on (G,U), then*

$$\sum_{u\in U} f^+(u) = 0.$$

Proof. By simple counting, we get

$$\begin{aligned}\sum_{u\in U} f^+(u) &= \sum_{v\in V_G\smallsetminus U} f^-(v) - \sum_{v\in V_G\smallsetminus U} f^+(v)\\ &= \sum_{v\in V_G\smallsetminus U} \big(f^-(v) - f^+(v)\big) = 0.\end{aligned}$$

The first equality is from the fact that there is no arc having head in any terminal, the latter from the definition of a flow on a network. □

Furthermore, $f^+(u) \neq 0$ for each terminal u, therefore, $f^+(U)$ belongs to set

$$S_n = \{(s_1,\dots,s_n)\colon s_1,\dots,s_n \in \mathbb{Z}_5 \smallsetminus \{0\}, s_1+\dots+s_n = 0\}.$$

For each $s \in S_n$, let $F_{G,U}(s)$ be the number of flows on (G,U) such that $f^+(U) = s$.

Let $P = \{Q_1,\dots,Q_r\}$ be a partition of the set $\{1,\dots,n\}$. We call P *proper* if each of $Q_1,\dots,Q_r$ contains at least two elements and we denote the set of all proper partitions of $\{1,\dots,n\}$ by $\mathcal{P}_n = \{P_1,\dots,P_{p_n}\}$.

For $s = (s_1,\dots,s_n) \in S_n$ and $P = \{Q_1,\dots,Q_r\} \in \mathcal{P}_n$, we define a *compatibility* of s and P as

$$\chi(s,P) = \begin{cases} 1 & \text{if } \sum_{i\in Q_j} s_i = 0 \text{ for each } j \in \{1,\dots,r\},\\ 0 & \text{otherwise.}\end{cases}$$

Furthermore, for $s \in S_n$, let $\chi_n(s) = \big(\chi(s,P_1),\dots,\chi(s,P_{p_n})\big)$.

Using Tutte's contraction/deletion formula [12], the following lemma can be proved. For more details see [6].

Lemma 2.3. *Let (G,U) be a simple network with n terminals. Then there exist integers $x_1,\dots,x_{p_n}$ such that $F_{G,U}(s) = \sum_{i=1}^{p_n} x_i\chi(s,P_i)$ for every $s \in S_n$.*

Example 2.4. Let C_n be a cycle with vertices $\{v_1,\dots,v_n\}$. Denote by $\widetilde{C_n}$ the graph obtained from C_n by adding a new vertex u_i and a new edge $\{v_i,u_i\}$ for each $i = 1,\dots,n$, see Figure 2.3. Clearly, $(\widetilde{C_n},U)$ where $U = \{u_1,\dots,u_n\}$ is a simple network. The graph $\widetilde{C_n}$ is sometimes called *n-sunlet*.

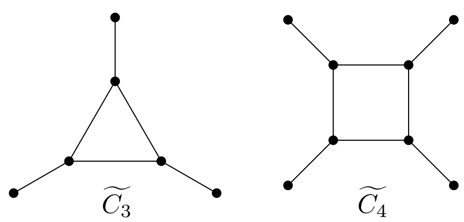

Figure 2.3: Examples of sunlets: $\widetilde{C_3}$ and $\widetilde{C_4}$.

For $n = 3$, there is only one proper partition, namely $P = \big\{\{1,2,3\}\big\}$, therefore by Lemma 2.3, there exists such x that $F_{\widetilde{C_3},U}(s) = x$ for each $s \in S_3$. In the proof of Theorem 2.1, we showed $x = 2$.

For $n = 4$, there are four proper partitions

$$P_1 = \{\{1,2,3,4\}\}, \qquad P_2 = \{\{1,2\},\{3,4\}\},$$
$$P_3 = \{\{1,3\},\{2,4\}\}, \qquad P_4 = \{\{1,4\},\{2,3\}\}$$

and the integers from Lemma 2.3 are $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 1$.

Table 2.1 contains all "types" of $s \in S_4$, their compatibilities with these partitions $\chi_4(s)$ and the numbers of flows $F_{\widetilde{C_4},U}(s)$ that can be easily determined directly. Other vectors from S_4 give the same results as some displayed in the table since there is no $s \in S_4$ compatible with all four proper partitions.

s	$\chi(s, P_1)$	$\chi(s, P_2)$	$\chi(s, P_3)$	$\chi(s, P_4)$	$F_{\widetilde{C_4},U}(s)$
$(1,4,4,1)$	1	1	1	0	2
$(1,4,1,4)$	1	1	0	1	3
$(1,4,2,3)$	1	1	0	0	2
$(1,1,4,4)$	1	0	1	1	2
$(1,2,4,3)$	1	0	1	0	1
$(1,2,3,4)$	1	0	0	1	2
$(1,1,1,2)$	1	0	0	0	1

Table 2.1: Examples of $s \in S_4$ and corresponding $\chi_4(s)$ and $F_{\widetilde{C_4},U}(s)$.

2.1.1 Forbidden Networks

First method. Let H be a graph such that $V = \{v_1, \ldots, v_n\}$ is the set of all vertices of degree 2 and all the other vertices have degree 3. Let $\widetilde{H}$ be a graph obtained from H by adding new vertices $U = \{u_1, \ldots, u_n\}$ and edges $\{u_i, v_i\}$ for each $i = 1, \ldots, n$. Clearly, the network $(\widetilde{H}, U)$ is simple.

Let $S_H = \left\{ s \in S_n \colon F_{\widetilde{H},U}(s) > 0 \right\}$. Denote by V_n and V_H the linear hulls of $\{\chi_n(s) \colon s \in S_n\}$ and $\{\chi_n(s) \colon s \in S_H\}$, respectively, both in $\mathbb{Q}^{p_n}$. As $S_H \subseteq S_n$, $V_H \subseteq V_n$.

Theorem 2.5 (Kochol 2006 [7])**.** *If $V_H = V_n$ then H cannot be a subgraph of any minimal counterexample to the 5-Flow Conjecture.*

Proof. Let G_m be some minimal counterexample to the 5-Flow Conjecture. As we already mentioned, G_m is a 3-connected, cubic graph. Let H be a subgraph of G_m having minimum degree 2 and $V = \{v_1, \ldots, v_n\}$, $U = \{u_1, \ldots, u_n\}$ and $\widetilde{H}$ be as defined above.

In case there is some edge $e \in E_{G_m} \setminus E_H$ with both ends in V, we subdivide the edge by a new vertex of degree 2 and denote by G the graph obtained after subdividing all such edges. Therefore, there exists a unique edge $e_i = \{v_i, v'_i\} \in E_G \setminus E_H$ in G for each $i = 1, \ldots, n$. Note that the edges $e_1, \ldots, e_n$ are pairwise different whereas the vertices $v'_1, \ldots, v'_n$ do not need to be such. Let $I = G[V_G \setminus V_H]$ and $\widetilde{I}$ be the graph obtained from I by adding new pairwise different vertices $W = \{w_1, \ldots, w_n\}$ and edges $\{v'_i, w_i\}$ for each $i = 1, \ldots, n$, see Figure 2.4.

Clearly, if there exists $s \in S_n$ such that both $F_{\widetilde{H},U}(s)$ and $F_{\widetilde{I},W}(s)$ are positive, i.e. there exist some flows f_H and f_I on simple networks $(\widetilde{H}, U)$ and $(\widetilde{I}, W)$,

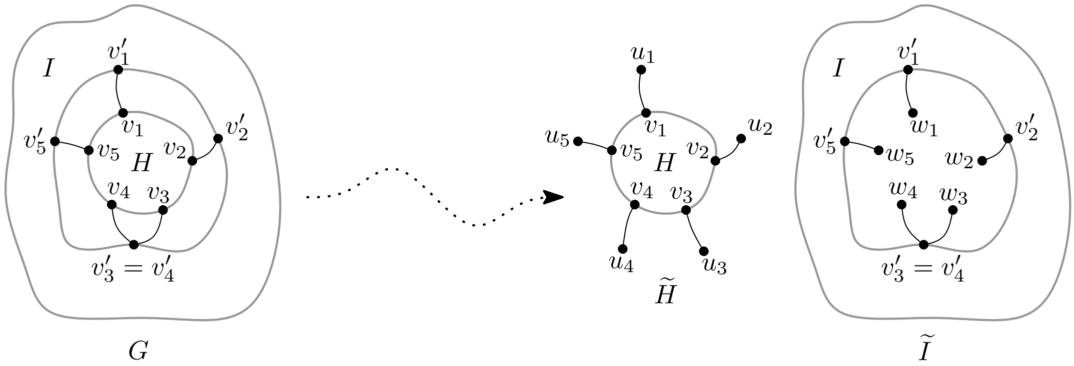

Figure 2.4: Graphs $\widetilde{H}$ and $\widetilde{I}$ obtained from a graph G.

respectively, such that $f_H^+(U) = f_I^+(W) = s$, then there exists a nowhere-zero 5-flow f_G on graph G obtained by "merging" flows f_H and $-f_I$. This flow can be easily transformed into a nowhere-zero 5-flow on graph G_m and, therefore, we get a contradiction. Thus, $F_{\widetilde{H},U}(s) \cdot F_{\widetilde{I},W}(s) = 0$ for each $s \in S_n$ and $F_{\widetilde{I},W}(s) = 0$ for each $s \in S_H$.

As we know from Lemma 2.3, there exist integers $x_1, \ldots, x_{p_n}$ such that the formula $F_{\widetilde{I},W}(s) = \sum_{i=1}^{p_n} x_i \chi(s, P_i)$ holds for each $s \in S_n$. Let $t_1, \ldots, t_r \in S_H$ be such that $\chi_n(t_1), \ldots, \chi_n(t_r)$ form a basis of vector space $V_H = V_n$. Then for each $s \in S_n$, there exist numbers $y_1, \ldots, y_r$ such that $\chi(s, P_i) = \sum_{j=1}^{r} y_j \chi(t_j, P_i)$ for each $i = 1, \ldots, p_n$ and

$$F_{\widetilde{I},W}(s) = \sum_{i=1}^{p_n} x_i \chi(s, P_i) = \sum_{i=1}^{p_n} x_i \left(\sum_{j=1}^{r} y_j \chi(t_j, P_i) \right) =$$
$$= \sum_{j=1}^{r} y_j \left(\sum_{i=1}^{p_n} x_i \chi(t_j, P_i) \right) = \sum_{j=1}^{r} y_j F_{\widetilde{I},W}(t_j) = 0,$$

where the last equality holds because $t_1, \ldots, t_r \in S_H$.

Let G_m/H be the graph obtained from G_m by contracting the whole subgraph H into one vertex. Clearly, G_m/H is smaller than G_m and bridgeless, therefore, G_m/H admits a nowhere-zero 5-flow. This flow can be transformed into a flow on $(\widetilde{I}, W)$ and, therefore, there exists $s \in S_n$ such that $F_{\widetilde{I},W}(s) > 0$.

As this is a contradiction, H cannot be a subgraph of a minimal counterexample to the 5-Flow Conjecture. $\square$

Second method. In 2010, Kochol [8] has introduced two modifications to this method – firstly, he replaced a network $(\widetilde{H}, U)$ by a smaller one to show that H cannot be a subgraph of any minimal counterexample to the 5-Flow Conjecture, and secondly, he introduced the use of a permutation group to reduce the size of computations. We discuss this second part later in Subsection 2.1.2.

Let H, $V = \{v_1, \ldots, v_n\}$, $\widetilde{H}$ and $U = \{u_1, \ldots, u_n\}$ be as mentioned in the first method. Further let H' be a graph and $V' = \{v'_1, \ldots, v'_n\}$ set (possibly multiset) of its vertices. We denote by $\widetilde{H'}$ the graph obtained from H' by adding vertices $U' = \{u'_1, \ldots, u'_n\}$ and edges $\{u'_i, v'_i\}$ where $i = 1, \ldots, n$. Similarly to S_H and V_H, let $S_{H'} = \left\{ s \in S_n \colon F_{\widetilde{H'},U}(s) > 0 \right\}$ and $V_{H'}$ be the linear hull of $\{\chi_n(s) \colon s \in S_{H'}\}$ in $\mathbb{Q}^{p_n}$.

Let G be a cubic graph such that H is its subgraph and let $\{v_i, v_i''\}$ be the unique edge of $E_G \setminus E_H$ with an end in vertex v_i for each $i = 1, \ldots, n$. Then we say that a graph G' is obtained from G by *replacing* H by H' if it arises from graphs $G[V_G \setminus V_H]$ and $\widetilde{H'}$ by identifying vertices v_i'' and u_i', see Figure 2.5.

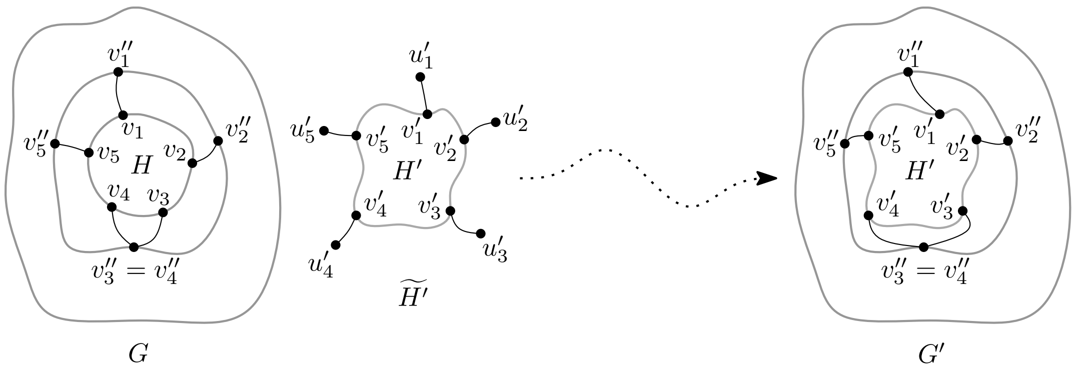

Figure 2.5: Graph G' obtained from G by replacing H by H'.

We say that H is *replaceable* by H' in a class $\mathcal{C}$ of graphs if the graph obtained from G by replacing H by H' is bridgeless for each graph G of $\mathcal{C}$.

Theorem 2.6 (Kochol 2010 [8]). *If there exists some graph H' smaller than H such that H is replaceable by H' in the class of cyclically 6-connected graphs and $V_{H'} \subseteq V_H$ then H cannot be a subgraph of any minimal counterexample to the 5-Flow Conjecture.*

Proof. Suppose that G_m is a minimal counterexample to the 5-Flow Conjecture and H is its subgraph such that its minimum degree is 2. Then let $V = \{v_1, \ldots, v_n\}$, $U = \{u_1, \ldots, u_n\}$ and $\widetilde{H}$ be as mentioned in the proof of Theorem 2.5. Similarly, let H' be as assumed and $V' = \{v_1', \ldots, v_n'\}$, $U' = \{u_1', \ldots, u_n'\}$ and $\widetilde{H'}$ as described above. As Kochol [6] proved, G_m is cyclically 6-connected.

Let G be a graph obtained from G_m by subdividing all edges of $E_{G_m} \setminus E_H$ with both ends in V. Finally, let I be the "rest" of the graph G after removing subgraph H and $(\widetilde{I}, W)$ be the network obtained from I by adding n new vertices and edges, see the proof of Theorem 2.5 and Figure 2.4.

If there exists such $s \in S_n$ that $F_{\widetilde{H},U}(s), F_{\widetilde{I},W}(s) > 0$ then there exist flows f_H and f_I on networks $(\widetilde{H}, U)$ and $(\widetilde{I}, W)$, respectively, where $f_H^+(U) = f_I^+(W) = s$. Therefore, the flow obtained by "merging" flows f_H and $-f_I$ is a nowhere-zero 5-flow on graph G, which is a contradiction. Therefore, at least one of $F_{\widetilde{H},U}(s)$ and $F_{\widetilde{I},W}(s)$ must be 0 for each $s \in S_n$ and $F_{\widetilde{I},W}(s) = 0$ for each $s \in S_H$.

As stated in Lemma 2.3, there exist integers $x_1, \ldots, x_{p_n}$ such that formula $F_{\widetilde{I},W}(s) = \sum_{i=1}^{p_n} x_i \chi(s, P_i)$ holds for each $s \in S_n$. Let $\chi_n(t_1), \ldots, \chi_n(t_r)$ for some $t_1, \ldots, t_r \in S_H$ be a basis of vector space V_H. As already mentioned, $F_{\widetilde{I},W}(t_j) = 0$ for each $j = 1, \ldots, r$.

Since $V_{H'} \subseteq V_H$, there exist some numbers $y_1, \ldots, y_r$ for each $s \in S_{H'}$ such that $\chi(s, P_i) = \sum_{j=1}^{r} y_j \chi(t_j, P_i)$ holds for each $i = 1, \ldots, p_n$ and

$$F_{\widetilde{I},W}(s) = \sum_{i=1}^{p_n} x_i \chi(s, P_i) = \sum_{i=1}^{p_n} x_i \left(\sum_{j=1}^{r} y_j \chi(t_j, P_i) \right) =$$

$$= \sum_{j=1}^{r} y_j \left(\sum_{i=1}^{p_n} x_i \chi(t_j, P_i) \right) = \sum_{j=1}^{r} y_j F_{\widetilde{I},W}(t_j) = 0.$$

Let G' be the graph obtained from G by replacing H by H'. By an assumption that H' is smaller than H and H is replaceable by H' in the class of cyclically 6-connected graphs, G' is smaller than G and still bridgeless. Thus, there exists a nowhere-zero 5-flow on G' and $s \in S_{H'}$ such that $F_{\widetilde{I},W}(s) > 0$.

Therefore, we get a contradiction and H cannot be a subgraph of any minimal counterexample to the 5-Flow Conjecture. □

The comparison of these two methods. Note that Theorem 2.5 is a special case of Theorem 2.6 for H' consisting of only one vertex, since $V_n = V_{H'}$ holds for such H'.

The main advantage of the method presented in Theorem 2.6 is that we can select the graph H' such that there do not exist many flows on the network $(\widetilde{H'}, U')$. As a consequence of such choice, the dimension of the vector space $V_{H'}$ would be small and it would be easier to verify that $V_{H'} \subseteq V_H$.

2.1.2 The Computations

We can formulate the problem of determining whether $V_H = V_n$ in Theorem 2.5 in terms of matrices: Let M_n and M_H be the matrices where rows are exactly $\chi_n(s)$ for $s \in S_n$ and $s \in S_H$, respectively. Then $V_H = V_n$ if and only if the ranks of M_H and M_n are equal.

Let $\mathcal{A}$ be the automorphism group of $\mathbb{Z}_5$, i.e. its elements are

$$\alpha_1 = \mathrm{id}, \qquad \alpha_2 = (1, 2, 4, 3),$$
$$\alpha_3 = (1, 3, 4, 2), \qquad \alpha_4 = (1, 4)(2, 3).$$

We can note that $\alpha_i(x) = x \cdot i$ in $\mathbb{Z}_5$.

For $s = (s_1, \ldots, s_n) \in S_n$ and $\alpha \in \mathcal{A}$, let $\alpha(s) = (\alpha(s_1), \ldots, \alpha(s_n))$. Clearly, $F_{G,U}(s) = F_{G,U}(\alpha(s))$ for any simple network (G, U) and $\chi_n(s) = \chi_n(\alpha(s))$. Therefore, we can divide all elements of S_n into classes

$$\Sigma_n = \big\{\{\alpha_1(s), \alpha_2(s), \alpha_3(s), \alpha_4(s)\} \colon s \in S_n\big\}$$

such that if $\sigma \in \Sigma_n$ then $\chi_n(s)$ is used as a row in matrix M_n or M_H either for all $s \in \sigma$ or for none of them. Moreover, in the first case, it would be the same row used four times.

Therefore, it is sufficient to use only one representative of each $\sigma \in \Sigma_n$, without loss of generality the one with $s_1 = 1$. By this operation, we can reduce the numbers of rows in both matrices to one quarter of their original numbers.

Kochol [7] used the method described in Theorem 2.5 with C_5, C_6, C_7 and C_8 in the role of H and computed the ranks of the matrices using computer, see

H	size of M_n	size of M_H	rank M_n	rank M_H	Note
C_5	51×11	45×11	11	11	[7]
C_6	205×41	151×41	40	40	[7]
C_7	819×162	483×162	147	147	[7]
C_8	$3\,277 \times 715$	$1\,513 \times 715$	568	568	[7]
C_9	$13\,107 \times 3\,425$	$4\,665 \times 3\,425$	2 227	2 227	

Table 2.2: The sizes and ranks of some matrices M_n and M_H from Theorem 2.5.

Table 2.2. Since he has not provided the source code he used and also to verify the result independently, we have created a new program in Sage [11] that uses the same method and has validated Kochol's results. Moreover, it has computed the rank for the case of $H = C_9$, see Table 2.2.

We can use the method of counting ranks of matrices also to verify the condition $V_{H'} \subseteq V_H$ in Theorem 2.6. In this case, it would be M_H and $M_{H'}$ containing $\chi_n(s)$ as a row if and only if $s \in S_H$ and $s \in S_H \cup S_{H'}$, respectively. Again, it is possible to use the automorphism group $\mathcal{A}$ and the assumption that $s_1 = 1$ for all used $s = (s_1, \ldots, s_n) \in S_n$.

Kochol [8] used this method with C_9 and C_{10} as a graph H together with shorter cycles C_7 and C_8, respectively, and one isolated edge as a graph H', see Figure 2.6.

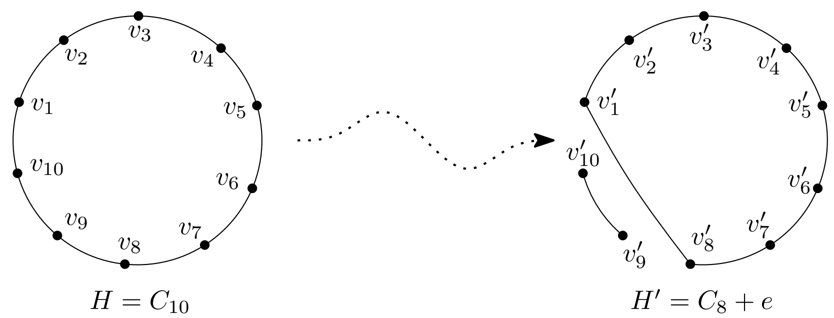

Figure 2.6: Example of H and H' from Theorem 2.6 used by Kochol [8].

The reduction of the size of matrices using a permutation group. In 2010, Kochol [8] also provided a method using a permutation group to reduce the size of computed matrices.

Let us remind (page 8) that

$$S_n = \{(s_1, \ldots, s_n) \colon s_1, \ldots, s_n \in \mathbb{Z}_5 \setminus \{0\}, s_1 + \cdots + s_n = 0\},$$

$\mathcal{P}_n$ is the set of all proper partitions of $\{1, \ldots, n\}$, i.e. partitions $P = \{Q_1, \ldots, Q_r\}$ where each $Q \in P$ contains at least two elements,

$$\chi(s, P) = \begin{cases} 1 & \text{if } \sum_{i \in Q} s_i = 0 \text{ for each } Q \in P, \\ 0 & \text{otherwise,} \end{cases}$$

and $\chi_n(s) = \big(\chi(s, P_1), \ldots, \chi(s, P_{p_n})\big)$.

Let Γ be a permutation group on $\{1,\dots,n\}$, i.e. a subgroup of the group of all permutations of elements $1,\dots,n$. For $\gamma \in \Gamma$ and $Q \subseteq \{1,\dots,n\}$, let $\gamma(Q) = \{\gamma(q) \colon q \in Q\}$, and similarly, $\gamma(P) = \{\gamma(Q) \colon Q \in P\}$ for any $P \in \mathcal{P}_n$.

Denote by $\boldsymbol{\mathcal{P}}_n$ the partition of $\mathcal{P}_n$ into classes

$$\boldsymbol{\mathcal{P}}_n = \{\boldsymbol{P}_1,\dots,\boldsymbol{P}_{\boldsymbol{p}_n}\} = \{\{\gamma(P) \colon \gamma \in \Gamma\} \colon P \in \mathcal{P}_n\}.$$

Then for each $\boldsymbol{P} \in \boldsymbol{\mathcal{P}}_n$ and $s \in S_n$, denote $\boldsymbol{\chi}(s,\boldsymbol{P}) = \sum_{P\in\boldsymbol{P}} \chi(s,P)$ and $\boldsymbol{\chi}_n(s) = \big(\boldsymbol{\chi}(s,\boldsymbol{P}_1),\dots,\boldsymbol{\chi}(s,\boldsymbol{P}_{\boldsymbol{p}_n})\big)$.

Let $\gamma \in \Gamma$ and $s = (s_1,\dots,s_n) \in S_n$, then we define $\gamma(s) = (s_{\gamma(1)},\dots,s_{\gamma(n)})$. Since $\chi(s,P) = \chi(\gamma(s),\gamma^{-1}(P))$ for each $P \in \mathcal{P}_n$ and $\gamma \in \Gamma$, the formula $\boldsymbol{\chi}(s,\boldsymbol{P}) = \boldsymbol{\chi}(\gamma(s),\boldsymbol{P})$ holds for each $\boldsymbol{P} \in \boldsymbol{\mathcal{P}}_n$.

Therefore, $\boldsymbol{\chi}_n(s) = \boldsymbol{\chi}_n(\gamma(s))$ for each $s \in S_n$ and $\gamma \in \Gamma$ and we can divide the set S_n into the classes

$$\boldsymbol{S}_n = \{\{\gamma(s) \colon \gamma \in \Gamma\} \colon s \in S_n\}.$$

For $\boldsymbol{s} \in \boldsymbol{S}_n$ and $\boldsymbol{P} \in \boldsymbol{\mathcal{P}}_n$, we define $\boldsymbol{\chi}(\boldsymbol{s},\boldsymbol{P}) = \boldsymbol{\chi}(s,\boldsymbol{P})$ and $\boldsymbol{\chi}_n(\boldsymbol{s}) = \boldsymbol{\chi}_n(s)$ where $s \in \boldsymbol{s}$ is arbitrary.

Finally for each $\boldsymbol{s} \in \boldsymbol{S}_n$, we define $\boldsymbol{F}_{G,U}(\boldsymbol{s}) = \sum_{\gamma\in\Gamma} F_{G,U}(\gamma(s))$ where $s \in \boldsymbol{s}$ is again arbitrary.

Let us formulate the following Lemma similar to Lemma 2.3. It can be proved by a straightforward computation and using the definitions above, see [8].

Lemma 2.7. *Let (G,U) be a simple network with n terminals and Γ be a permutation group on $\{1,\dots,n\}$. Then there exist integers $\boldsymbol{x}_1,\dots,\boldsymbol{x}_{\boldsymbol{p}_n}$ such that $\boldsymbol{F}_{G,U}(\boldsymbol{s}) = \sum_{i=1}^{\boldsymbol{p}_n} \boldsymbol{x}_i \boldsymbol{\chi}(\boldsymbol{s},\boldsymbol{P}_i)$ for every $\boldsymbol{s} \in \boldsymbol{S}_n$.*

Let $(\widetilde{H},U)$ and $(\widetilde{H'},U')$ be simple networks with n terminals and Γ be a permutation group on $\{1,\dots,n\}$, denote by

$$\boldsymbol{S}_H = \{\boldsymbol{s} \in \boldsymbol{S}_n \colon \boldsymbol{F}_{\widetilde{H},U}(\boldsymbol{s}) > 0\}, \qquad \boldsymbol{S}_{H'} = \{\boldsymbol{s} \in \boldsymbol{S}_n \colon \boldsymbol{F}_{\widetilde{H'},U}(\boldsymbol{s}) > 0\}$$

and by $\boldsymbol{V}_H$ and $\boldsymbol{V}_{H'}$ the linear hulls of $\{\boldsymbol{\chi}_n(\boldsymbol{s}) \colon \boldsymbol{s} \in \boldsymbol{S}_H\}$ and $\{\boldsymbol{\chi}_n(\boldsymbol{s}) \colon \boldsymbol{s} \in \boldsymbol{S}_{H'}\}$, respectively, both in $\mathbb{Q}^{\boldsymbol{p}_n}$.

We say that Γ *acts regularly* on a simple network (G,U) with $U = \{u_1,\dots,u_n\}$ if for each $\gamma \in \Gamma$, there exists an automorphism φ of G such that $\varphi(u_i) = u_{\gamma(i)}$ for each $i = 1,\dots,n$.

Lemma 2.8. *Let (G,U) be a simple network with n terminals and Γ be a permutation group that acts regularly on (G,U). Then $F_{G,U}(s) = F_{G,U}(\gamma(s))$ for each $s \in S_n$ and $\gamma \in \Gamma$.*

Again, the proof is quite straightforward and can be found in [8].

Now, we can formulate a result similar to Theorem 2.6 extended by using a permutation group Γ.

Theorem 2.9 (Kochol 2010 [8])**.** *Let Γ be a permutation group on $\{1,\dots,n\}$ that acts regularly on $(\widetilde{H},U)$. If there exists some graph H' smaller than H such that H is replaceable by H' in the class of cyclically 6-connected graphs and $V_{H'} \subseteq V_H$, then H cannot be a subgraph of any minimal counterexample to the 5-Flow Conjecture.*

Proof. The proof begins exactly the same way as the proof of Theorem 2.6. Let graphs G_m, G, H, H' and I and networks $(\widetilde{H}, U)$, $(\widetilde{H'}, U')$ and $(\widetilde{I}, W)$ be as defined in the proof of Theorem 2.6. Therefore, $F_{\widetilde{H},U}(s)F_{\widetilde{I},W}(s) = 0$ for each $s \in S_n$.

By Lemma 2.8, $F_{\widetilde{H},U}(s) = F_{\widetilde{H},U}(\gamma(s))$ for each $s \in S_n$ and $\gamma \in \Gamma$. Therefore, $\boldsymbol{s} \in \boldsymbol{S}_H$ if and only if $F_{\widetilde{H},U}(s) > 0$ holds for every $s \in \boldsymbol{s}$. Consequently, $F_{\widetilde{I},W}(s) = 0$ for every $s \in \boldsymbol{s} \in \boldsymbol{S}_H$ and $\boldsymbol{F}_{\widetilde{I},W}(\boldsymbol{s}) = 0$ for each $\boldsymbol{s} \in \boldsymbol{S}_H$.

We already know from Lemma 2.7 that there exist such integers $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\boldsymbol{p}_n}$ that $\boldsymbol{F}_{G,U}(\boldsymbol{s}) = \sum_{i=1}^{\boldsymbol{p}_n} \boldsymbol{x}_i \boldsymbol{\chi}(\boldsymbol{s}, \boldsymbol{P}_i)$ holds for every $\boldsymbol{s} \in \boldsymbol{S}_n$. Let $\boldsymbol{\chi}_1, \ldots, \boldsymbol{\chi}_r \in \boldsymbol{S}_H$ such that $\boldsymbol{\chi}_n(\boldsymbol{t}_1), \ldots, \boldsymbol{\chi}_n(\boldsymbol{t}_n)$ is a basis of the vector space $\boldsymbol{V}_H$.

By assumption, $\boldsymbol{V}_{H'} \subseteq \boldsymbol{V}_H$, and therefore, there exist some numbers $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_r$ for every $\boldsymbol{s} \in \boldsymbol{S}_{H'}$ such that $\boldsymbol{\chi}(\boldsymbol{s}, \boldsymbol{P}_i) = \sum_{j=1}^{r} \boldsymbol{y}_j \boldsymbol{\chi}(\boldsymbol{t}_j, \boldsymbol{P}_i)$ for each $i = 1, \ldots, \boldsymbol{p}_n$ and

$$\begin{aligned}\boldsymbol{F}_{\widetilde{I},W}(\boldsymbol{s}) = \sum_{i=1}^{\boldsymbol{p}_n} \boldsymbol{x}_i \boldsymbol{\chi}(\boldsymbol{s}, \boldsymbol{P}_i) &= \sum_{i=1}^{\boldsymbol{p}_n} \boldsymbol{x}_i \left(\sum_{j=1}^{r} \boldsymbol{y}_j \boldsymbol{\chi}(\boldsymbol{t}_j, \boldsymbol{P}_i) \right) = \\ &= \sum_{j=1}^{r} \boldsymbol{y}_j \left(\sum_{i=1}^{\boldsymbol{p}_n} \boldsymbol{x}_i \boldsymbol{\chi}(\boldsymbol{t}_j, \boldsymbol{P}_i) \right) = \sum_{j=1}^{r} \boldsymbol{y}_j \boldsymbol{F}_{\widetilde{I},W}(\boldsymbol{t}_j) = 0.\end{aligned}$$

Denote by G' the graph obtained from G by replacing H by H'. Since H is replaceable by a smaller H', G' is smaller than G and bridgeless and, therefore, there exists a nowhere-zero flow on G' and $s \in S_{H'}$ such that $F_{\widetilde{I},W}(s) > 0$. Then there exists $\boldsymbol{s} \in \boldsymbol{S}_{H'}$ such that $s \in \boldsymbol{s}$ and $\boldsymbol{F}_{\widetilde{I},W}(\boldsymbol{s}) > 0$, which is a contradiction.

Therefore H cannot be a subgraph of any minimal counterexample to the 5-Flow Conjecture. □

As already mentioned, Kochol [8] tried the method from Theorem 2.6 using C_9 and C_{10} as graph H. At the same time, he also applied this reduction of the sizes of the matrices. As a permutation group Γ, he used dihedral groups D_9 and D_{10}, respectively, i.e. the groups of all symmetries of the circuits. Table 2.3 presents the sizes and ranks of matrices computed by Kochol and validated by our program and one more row for our new result for $H = C_{11}$. The computation time for this result was about 5 days.

H	size of M_H	size of $M_{H'}$	rank M_H	rank $M_{H'}$	Note
C_8	122×81	176×81	62	62	
C_9	262×238	430×238	151	151	[8]
C_{10}	$792 \times 1\,079$	$1\,415 \times 1\,079$	539	539	[8]
C_{11}	$1\,972 \times 4\,752$	$3\,937 \times 4\,752$	1 699	1 699	

Table 2.3: The sizes and ranks of some matrices M_H and $M_{H'}$ from Theorem 2.6.

Since ranks of $M_{C_{11}}$ and $M_{C'_{11}}$ are equal and from Theorem 2.6, this proves the following Theorem.

Theorem 2.10. *Any minimal counterexample to the 5-Flow Conjecture does not contain any circuit of length less than 12.*

Some remarks on implementation issues. There are many possible ways how to implement the test whether some vector space is a subspace of another. Sage provides class `VectorSpace` where we can for each vector check whether it is or is not part of the space. This method is quite inefficient as there is no way how to add new vector into an existing vector space, therefore, new instance of `VectorSpace` must be created in order to extent the vector space.

On the other hand, Sage also provides class `Matrix` that can be used to compute the ranks as mentioned above. Its function `rank()` is quite fast even used on rather large matrices. However, there must be the whole matrix in the memory, therefore, for even larger matrices, we provide computing of the rank using "buffered" matrix: if the number of rows exceed upper limit, LU decomposition of the matrix is calculated and only pivoting rows are stored for determining the rank, see Code snippet 2.1.

```
if rn >= Buffer:
  M = Matrix(QQ, Mn)
  rn = M.rank()
  P, L, U = M.LU()
  Mn = [ U[i].list() for i in range(rn) ]
```

Code snippet 2.1: LU decomposition after matrix buffer overflow.

Furthermore, in order to optimize the run of the program, it is necessary to organize the code in such way that there are not repeatedly computed the same structures, e.g. both Code snippet 2.2 and Code snippet 2.3 are generating the same list of all n-tuples in the same order. In the first case, each $(n-1)$-tuple is generated once and extended four times, whereas in the second case, each $(n-1)$-tuple is generated every time we want to extend it, i.e. four times. We tried a few tests to compare the time of executing these snippets and in most of the cases the first one was slightly faster.

```
for t in NTuples(n-1):
  for i in [1, 2, 3, 4]:
    yield t + [i]
```

Code snippet 2.2: First way of generating n-tuples.

```
for i in [1, 2, 3, 4]:
  for t in NTuples(n-1):
    yield [i] + t
```

Code snippet 2.3: Second way of generating n-tuples.

2.2 Modifications of the Approach

We have further studied the cases where an even circuit $H = C_{2k}$ is replaced by a non-crossing perfect matching of its vertices as a graph H', see Figure 2.7.

We have to be more careful because there might not be one universal matching H' such that H can be replaced by H' in an arbitrary 3-connected cubic graph without creating a bridge.

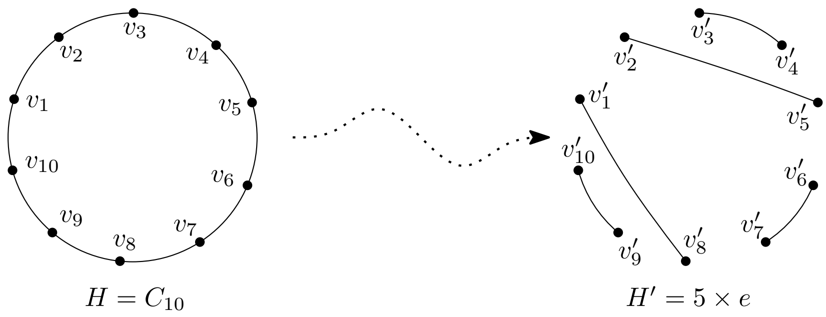

Figure 2.7: One example of possible non-crossing perfect matching used as graph H'.

Definition 2.11 ([13, Definition A.5.1, page 296]). Let G be a graph and v be a vertex of G and $F \subset E_G(v)$. The graph $G_{[v;F]}$ is obtained from G by splitting the edges of F away from v, i.e. adding a new vertex v' and changing the end v of the edges of F to be v'.

Theorem 2.12 (Vertex Splitting Lemma [13, Theorem A.5.2, page 296]). *Let G be a connected bridgeless graph, $v \in V_G$ (with $d(v) \geq 4$), and $e_0, e_1, e_2 \in E_G(v)$. Then either $G_{[v;\{e_0,e_1\}]}$ or $G_{[v;\{e_0,e_2\}]}$ is connected and bridgeless unless $G_{[v;\{e_0,e_1,e_2\}]}$ is not connected, i.e. $\{e_0, e_1, e_2\}$ is edge-cut.*

Theorem 2.13. *Let G be a 3-connected cubic graph containing $H = C_{2k}$ ($k > 1$) as a subgraph. Then there exists some non-crossing perfect matching H' such that replacing H by H' in a graph G does not create any bridge.*

Proof. We will proceed by induction.

For $k = 2$, let e_1, e_2, e_3, e_4 be the edges of G neighboring $H = C_4$ in clockwise direction and G/H be the graph obtained from G by contracting all edges of H into new vertex v. If $G/H_{[v;\{e_1,e_2,e_4\}]}$ is not connected then e_3 is a bridge in G/H and also in G, which is a contradiction. Therefore by Theorem 2.12, either $G/H_{[v;\{e_1,e_2\}]}$ or $G/H_{[v;\{e_1,e_4\}]}$ is connected and bridgeless. In both cases, decontraction of corresponding edges of H obtains non-crossing perfect matching.

For $k > 2$, let $e_1, \dots, e_{2k}$ be the edges of G around H in clockwise direction, v_i be the end of e_i in H for $i = 1, \dots, 2k$ and G/H be the graph obtained by contracting H into vertex v. In case that $G/H_{[v;\{e_1,e_2,e_{2k}\}]}$ is not connected, the graph obtained from G by deleting the edges $\{v_2, v_3\}$ and $\{v_{2k-1}, v_{2k}\}$ is not connected, which is a contradiction as G is 3-connected.

Therefore by Theorem 2.12, either $G/H_{[v;\{e_1,e_2\}]}$ or $G/H_{[v;\{e_1,e_{2k}\}]}$ is connected and bridgeless. In the first case, we decontract all edges of H except of $\{v_2, v_3\}$ and $\{v_{2k}, v_1\}$ and add a new edge $\{v_{2k}, v_3\}$ to obtain a graph G'. In the second case, we decontract all edges except of $\{v_1, v_2\}$ and $\{v_{2k-1}, v_{2k}\}$ and add a new edge $\{v_{2k-1}, v_2\}$ to obtain G'. In both case, G' can be easily transformed to a 3-connected graph where vertices $v_3, \dots, v_{2k-1}$ together with either v_{2k} or v_2 form a circuit of length $2(k-1)$. From the induction assumption, there exists non-crossing perfect matching H'' such that the replacement does not create a bridge. This graph together with either $\{e_1, e_2\}$ or $\{e_{2k}, e_1\}$ obtains graph H'. ☐

As a corollary, we have to verify that H can be replaced by any non-crossing perfect matching. This can be done in similar way as in Subsection 2.1.2: $\chi(s)$ (or $\boldsymbol{\chi}(\boldsymbol{s})$) is a row of $M_{H'}$ if and only if there exists non-crossing perfect matching compatible with s (or $s \in \boldsymbol{s}$, respectively).

Table 2.4 provides the comparison of the sizes of $M_{H'}$ for our selection of H' (marked as "pairs", see Figure 2.7) and the one selected by Kochol where one edge is isolated (marked as "1 edge", see Figure 2.6). The program for C_8 run less than a minute, whereas it was about five hours for C_{10}.

H	H'	size of M_H	size of $M_{H'}$	rank M_H	rank $M_{H'}$
C_8	pairs	122×81	149×81	62	62
C_8	1 edge	122×81	176×81	62	62
C_{10}	pairs	$792 \times 1\,079$	$1\,129 \times 1\,079$	539	539
C_{10}	1 edge	$792 \times 1\,079$	$1\,415 \times 1\,079$	539	539

Table 2.4: The comparison of sizes of the matrices for two possible graphs H'.

In case of an odd circuit as H, we can end up with a non-crossing matching and a triangle. We have not studied this case yet.

Conclusion

In this Thesis, we have presented a comprehensive view on the method introduced by Kochol in 2006 and improved in 2010 [7, 8].

Since Kochol did not share his implementation, we have also created program that has validated his results. Moreover, we have proved that any minimal counterexample to the 5-Flow Conjecture does not contain any circuit of length less than 12. This extends Kochol's result by excluding C_{11}. The source code of the program is provided.

The Future Work

Further work in this area can take some of the following directions:

- optimizing the source code and possibly using a combination of other programming languages to compute even larger matrices,
- studying more possible graphs in the role of H or H' in order to exclude some families of graphs from minimal counterexamples,
- studying some other ways to reduce the size of computed matrices,
- using the method on other open problems, e.g. other Tutte's conjectures.

Bibliography

[1] Appel, K. I. and Haken, W. Every planar map is four colorable. *Bulletin of the American Mathematical Society*, **82** (5), 711–712, 1976. doi:10.1090/S0002-9904-1976-14122-5.

[2] Appel, K. I. and Haken, W. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, **21** (3), 429–490, 1977. Available from: http://projecteuclid.org/euclid.ijm/1256049011.

[3] Appel, K. I., Haken, W., and Koch, J. Every planar map is four colorable. Part II: Reducibility. *Illinois Journal of Mathematics*, **21** (3), 491–567, 1977. Available from: http://projecteuclid.org/euclid.ijm/1256049012.

[4] Celmins, U. A. *On cubic graphs that do not have an edge 3-coloring*. Ph.D. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada, 1984.

[5] Jaeger, F. Flows and generalized coloring theorems in graphs. *Journal of Combinatorial Theory, Series B*, **26** (2), 205–216, 1979. doi:10.1016/0095-8956(79)90057-1.

[6] Kochol, M. Reduction of the 5-flow conjecture to cyclically 6-edge-connected snarks. *Journal of Combinatorial Theory, Series B*, **90** (1), 139–145, 2004. doi:10.1016/s0095-8956(03)00080-7.

[7] Kochol, M. Restrictions on smallest counterexamples to the 5-flow conjecture. *Combinatorica*, **26** (1), 83–89, 2006. doi:10.1007/s00493-006-0006-1.

[8] Kochol, M. Smallest counterexample to the 5-flow conjecture has girth at least eleven. *Journal of Combinatorial Theory, Series B*, **100** (4), 381–389, 2010. doi:10.1016/j.jctb.2009.12.001.

[9] Robbins, H. E. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, **46** (5), 281–283, 1939. doi:10.2307/2303897.

[10] Seymour, P. D. Nowhere-zero 6-flows. *Journal of Combinatorial Theory, Series B*, **30** (2), 130–135, 1981. doi:10.1016/0095-8956(81)90058-7.

[11] Stein, W. A. et al. *Sage Mathematics Software (Version 5.9)*. The Sage Development Team, 2013. Available from: http://www.sagemath.org.

[12] TUTTE, W. T. A contribution to the theory of chromatic polynomials. *Canadian Journal of Mathematics*, **6** (1), 80–91, 1954. doi:10.4153/cjm-1954-010-9.

[13] ZHANG, C.-Q. *Integer Flows and Cycle Covers of Graphs*. No. 205 in Monographs and Textbooks in Pure and Applied Mathematics. Marcel Dekker, Inc., New York, 1997. ISBN 0-8247-9790-6.

Lists of Figures, Tables and Code Snippets

List of Figures

1.1 An example of a pair of dual graphs. 3
1.2 The definition of an orientation $\vec{G}$ and a mapping f. 4
2.1 A contraction of C_3. 6
2.2 Flow in the neighborhood of C_3 before and after decontraction. . . 7
2.3 Examples of sunlets: $\widetilde{C_3}$ and $\widetilde{C_4}$. 8
2.4 Graphs $\widetilde{H}$ and $\widetilde{I}$ obtained from a graph G. 10
2.5 Graph G' obtained from G by replacing H by H'. 11
2.6 Example of H and H' from Theorem 2.6 used by Kochol. 13
2.7 One example of possible non-crossing perfect matching. 17

List of Tables

2.1 Examples of $s \in S_4$ and corresponding $\chi_4(s)$ and $F_{\widetilde{C_4},U}(s)$. 9
2.2 The sizes and ranks of some matrices M_n and M_H. 13
2.3 The sizes and ranks of some matrices M_H and $M_{H'}$. 15
2.4 The comparison of sizes of the matrices for two possible graphs H'. 18

List of Code Snippets

2.1 LU decomposition after matrix buffer overflow. 16
2.2 First way of generating n-tuples. 16
2.3 Second way of generating n-tuples. 16

List of Notation

C_n	circuit with n vertices
$d_G(v)$	degree of vertex v in graph G
E_G	set of the edges of the graph G
$E_G(v)$	set of the edges of the graph G with some end in v
$E^+_{\vec{G}}(v)$	set of the arcs of the graph $\vec{G}$ with tails in v
$E^-_{\vec{G}}(v)$	set of the arcs of the graph $\vec{G}$ with heads in v
$F_{G,U}(s)$	number of flows on (G,U) with flow s on terminals; see page 8
$\boldsymbol{F}_{G,U}(\boldsymbol{s})$	number of flows on (G,U) with flow $s \in \boldsymbol{s}$ on terminals; see page 14
$f^+(v)$	sum of all flow leaving vertex v
$f^-(v)$	sum of all flow entering vertex v
$G - e$	graph obtained from G by deleting edge e
G/H	graph obtained from G by contracting all edges of H into a new vertex
$G[V]$	subgraph of graph G induced by vertices V
$\widetilde{G}$	graph obtained from G by adding a distinct edge and vertex to each vertex of degree 2; see page 9
(G,U)	network with graph G and terminals U
$\mathcal{P}_n$	set of all proper partitions of $\{1,\ldots,n\}$; see page 8
$\boldsymbol{\mathcal{P}}_n$	set of all classes of proper partitions of $\{1,\ldots,n\}$ according to permutation group Γ; see page 14
S_n	set of all $(\mathbb{Z}_5 \setminus \{0\})^n$ vectors with sum 0; see page 8
$\boldsymbol{S}_n$	set of all classes of $(\mathbb{Z}_5 \setminus \{0\})^n$ vectors with sum 0 according to permutation group Γ; see page 14
V_G	set of the vertices of the graph G
$\mathbb{Z}_5$	Abelian group on $\{0,1,2,3,4\}$
$\chi(s,P)$	compatibility of vector s and partition P; see page 8
$\boldsymbol{\chi}(\boldsymbol{s},\boldsymbol{P})$	compatibility of class $\boldsymbol{s}$ and class $\boldsymbol{P}$; see page 14
$\chi_n(s)$	compatibility vector of vector s; see page 8
$\boldsymbol{\chi}_n(\boldsymbol{s})$	compatibility vector of class $\boldsymbol{s}$; see page 14

APPENDICES

Appendix A

Source Code of the Programs

The source code that we provide here is the minimal working code. The full source code is available online on author's website http://kam.mff.cuni.cz/~korcsok/masterthesis/. The full code also provide special functions, such as:

- program prints progress messages on console,
- the matrices are automatically saved on disk and
- there is possibility to compute only one of the matrices (for the case of large matrices).

A.1 Kochol's Basic Method

This method uses only Theorem 2.5 with $H = C_n$.

```
def Partitions(List):
  """
    Generates all proper partitions of List.
  """
  if len(List)==0:
    return [ [] ]
  if len(List)==1:
    return [ ]

  Res = []

  for A in Subsets( List[:-1] ):
    if len(A)>0:
      M = List[:-1]
      for a in A:
        M.remove(a)
      AA = A.list() + [List[-1]]
      Res += [ r + [ AA ] for r in Partitions(M) ]

  return Res
# Partitions


def NTuples(n):
  """
    Generates all n-tuples of {1, 2, 3, 4}^n.
  """
  if n==0:
```

```
    yield []
    return

  for t in NTuples(n-1):
    for i in [1, 2, 3, 4]:
      yield t + [i]
# NTuples


def TerminalValues(n):
  """
    Generates all values of n terminals with
    - first item = 1,
    - sum of all items = 0.
  """
  if n<=1:
    yield []
    return

  for t in NTuples(n-2):
    T = [1] + t
    s = Integers(5)(0)
    for i in T:
      s += i
    if s!=0:
      yield T + [-s]
# TerminalValues


def Compatibility(Partition, TerminalValues):
  """
    Returns 1 if the partition is compatible with
    vales on terminals.
  """
  for Class in Partition:
    s = Integers(5)(0)
    for Index in Class:
      s += TerminalValues[Index]
    if s!=0:
      return 0

  return 1
# Compatibility


def Chi(Partitions, TerminalValues):
  """
    Returns the vector of compatibility.
  """
  return [Compatibility(P, TerminalValues) for P in Partitions]
# Chi
```

```python


def IsFlow(TerminalValues):
  """
    Returns True if there exists a flow
    with given values on terminals.
  """
  val = [ Integers(5)(i) for i in range(1,5) ]
  for Val in TerminalValues:
    val = [ v + Val for v in val if v + Val > 0 ]

  return len(val) > 0
# IsFlow


def Generate(n,Buffer):
  """
    Generates the matrices M_n and M_{C_n} and
    counts their ranks.
    If some matrix is too long the rank is computed
    by parts with matrix buffer of given size.
  """
  Parts = Partitions(range(n))
  Vals = TerminalValues(n)

  Mn = []
  MCn = []
  rn = 0
  rCn = 0

  for V in Vals:
    X = Chi(Parts, V)
    Mn += [ X ]
    rn += 1
    if rn >= Buffer:
      M = Matrix(QQ, Mn)
      rn = M.rank()
      P, L, U = M.LU()
      Mn = [ U[i].list() for i in range(rn) ]
    if IsFlow(V):
      MCn += [ X ]
      rCn += 1
      if rCn >= Buffer:
        M = Matrix(QQ, MCn)
        rCn = M.rank()
        P, L, U = M.LU()
        MCn = [ U[i].list() for i in range(rCn) ]

  Mn  = Matrix(Mn)
  MCn = Matrix(MCn)
  rn  = Mn.rank()
```

```
  rCn = MCn.rank()

  return [ n, rn, rCn ]
# Generate
```

A.2 Kochol's Advanced Method

This method uses Theorem 2.9 with $H = C_n$, H' as displayed on Figure 2.6 and $\Gamma = D_n$.

```
def Partitions(List):
  """
    Generates all proper partitions of List.
  """
  if len(List)==0:
    yield []
    return
  if len(List)==1:
    return

  for A in Subsets( List[:-1] ):
    if len(A) > 0:
      M = List[:-1]
      for a in A:
        M.remove(a)
      AA = A.list() + [List[-1]]
      for r in Partitions(M):
        yield r + [ AA ]
# Partitions


def Classes(Partition):
  """
    Returns a list determining the class
    where the partition belongs.
  """
  m = max([ max(p) for p in Partition])
  Res = range(m + 1)

  for P in Partition:
    m = min(P)
    for p in P:
      Res[p] = m

  return Res
# Classes


def IsMinPartition(Part, Gamma):
  """
    Returns True if given partition is
```

```
        lexicographically minimal in its class.
    """
    X = Classes(Part)

    for r in [ GammaPartition(g, Part) for g in Gamma]:
        x = Classes(r)
        if x < X:
            return False

    return True
# IsMinPartition


def GammaPartition(Gamma, Partition):
    """
        Applies permutation Gamma on Partition.
    """
    Res = Set([])

    for P in Partition:
        P_ = Set([Gamma[r]-1 for r in P])
        Res = Res.union(Set([P_]))

    return Res
# GammaPartition


def PartitionClasses(Partitions, Gamma):
    """
        Returns list of classes of partitions.
    """
    Res = []
    for Partition in Partitions:
        if IsMinPartition(Partition, Gamma):
            Res += [ Set([ GammaPartition(g, Partition) for g in
                Gamma ]) ]

    return Res
# PartitionClasses


def NTuples(n):
    """
        Generates all n-tuples of {1, 2, 3, 4}^n.
    """
    if n==0:
        yield []
        return

    for t in NTuples(n-1):
        for i in [1, 2, 3, 4]:
```

```
        yield t + [i]
# NTuples


def TerminalValues(n):
  """
    Generates all values of n terminals with
    - first item = 1,
    - sum of all items = 0.
  """
  if n<=1:
    yield []
    return

  for t in NTuples(n-2):
    h = [1] + t
    s = Integers(5)(0)
    for i in h:
      s += i
    if s!=0:
      yield h + [-s]
#  TerminalValues


def Compatibility(Partition, TerminalValues):
  """
    Returns 1 if the partition is compatible with
    vales on terminals.
  """
  for Class in Partition:
    s = Integers(5)(0)
    for Index in Class:
      s +=  TerminalValues[Index]
    if s!=0:
      return 0

  return 1
# Compatibility


def Chi(PartitionClasses, TerminalValues):
  """
    Returns the vector of compatibility.
  """
  return [sum(Compatibility(P, TerminalValues) for P in Part)
     for Part in PartitionClasses]
# Chi


def IsFlow(TerminalValues):
  """
```

```
    Returns True if there exists a flow
    with given values on terminals.
  """
  val = [ Integers(5)(i) for i in range(1,5) ]
  for Val in  TerminalValues:
    val = [ v + Val for v in val if v + Val > 0 ]

  return len(val) > 0
# IsFlow


def EncodeTerminalValues(Values):
  """
    Encodes values on terminals into one number.
  """
  if len(Values) == 1:
    return Integer(Values[-1] - 1)

  return 4 * EncodeTerminalValues(Values[:-1]) + Integer(Values[-1] - 1)
# EncodeTerminalValues


def GammaValues(Values, Gamma):
  """
    Applies permutation Gamma on Values.
  """
  Res = []
  for i in range(len(Values)):
    Res += [ Values[Gamma[i] - 1] ]

  s = 1/Integers(5)(Res[0])
  if s != 1:
    return [ i * s for i in Res]

  return Res
# GammaValues


def Generate(n, Buffer):
  """
    Generates the matrices M_n and M'_n and
    counts their ranks.
    If some matrix is too long the rank is computed
    by parts with matrix buffer of given size.
  """
  G = DihedralGroup(n)
  Gamma = [ g.tuple() for g in G ]

  PartCls = PartitionClasses(Partitions(range(n)), Gamma)
  PartSml = [ range(n-2), [ n-2, n-1 ] ]
```

```
Vals = TerminalValues(n)

Used = []
for i in range(4^(n-1)):
  Used += [ False ]

Mn = []
MMn = []
rn = 0
rrn = 0

for H in Vals:
  if Used[EncodeTerminalValues(H)]:
    continue

  Used[EncodeTerminalValues(H)] = True
  X = Chi(PartCls, H)

  if IsFlow(H):
    for G in Gamma:
      Used[EncodeTerminalValues(GammaValues(H, G))] = True

    Mn += [ X ]
    rn += 1
    if rn >= Buffer:
      M = Matrix(QQ, Mn)
      rn = M.rank()
      P, L, U = M.LU()
      Mn = [ U[i].list() for i in range(rn) ]

    MMn += [ X ]
    rrn += 1
    if rrn >= Buffer:
      M = Matrix(QQ, MMn)
      rrn = M.rank()
      P, L, U = M.LU()
      MMn = [ U[i].list() for i in range(rrn) ]

    continue

  if (Compatibility(PartSml, H) == 1) and IsFlow(H[:-2]):
    for G in Gamma:
      Used[EncodeTerminalValues(GammaValues(H, G))] = True

    MMn += [ X ]
    rrn += 1
    if rrn >= Buffer:
      M = Matrix(QQ, MMn)
      rrn = M.rank()
      P, L, U = M.LU()
```

```
243             MMn = [ U[i].list() for i in range(rrn) ]
244
245     Mn  = Matrix(Mn)
246     MMn = Matrix(MMn)
247     rn  = Mn.rank()
248     rrn = MMn.rank()
249
250     return [ n, rn, rrn ]
251 # Generate
```

A.3 Modificated Advanced Method

This is the modification discussed in Section 2.2 and it also uses Theorem 2.9 with $\Gamma = D_n$.

First 179 lines are the same as in Section A.2.

```
180 def IsMatching(TerminalValues):
181     """
182       Returns True if there exists a perfect matching
183       with given values on terminals.
184     """
185     l = len(TerminalValues)
186     if l == 0:
187       return True
188     if l == 1:
189       return False
190
191     x = Integers(5)(0)
192     for i in range(l-1):
193       if x + TerminalValues[i] + TerminalValues[i+1] == 0:
194         H = TerminalValues[:i] + TerminalValues[i+2:]
195         return IsMatching(H)
196     if x + TerminalValues[0] + TerminalValues[l-1] == 0:
197       return IsMatching(TerminalValues[1:l-2])
198     return False
199 # IsMatching
200
201
202 def Generate(n, Buffer):
203     """
204       Generates the matrices M_n and M'_n and
205       counts their ranks .
206       If some matrix is too long the rank is computed
207       by parts with matrix buffer of given size.
208     """
209     G = DihedralGroup(n)
210     Gamma = [ g.tuple() for g in G ]
211
212     PartCls = PartitionClasses(Partitions(range(n)), Gamma)
213
214     Vals = TerminalValues(n)
```

```
  Used = []
  for i in range(4^(n-1)):
    Used += [ False ]

  Mn = []
  MMn = []
  rn = 0
  rrn = 0

  for H in Vals:
    if Used[EncodeTerminalValues(H)]:
      continue

    Used[EncodeTerminalValues(H)] = True
    X = Chi(PartCls, H)

    if IsFlow(H):
      for G in Gamma:
        Used[EncodeTerminalValues(GammaValues(H, G))] = True

      Mn += [ X ]
      rn += 1
      if rn >= Buffer:
        M = Matrix(QQ, Mn)
        rn = M.rank()
        P, L, U = M.LU()
        Mn = [ U[i].list() for i in range(rn) ]

      MMn += [ X ]
      rrn += 1
      if rrn >= Buffer:
        M = Matrix(QQ, MMn)
        rrn = M.rank()
        P, L, U = M.LU()
        MMn = [ U[i].list() for i in range(rrn) ]

      continue

    if IsMatching(H):
      for G in Gamma:
        Used[EncodeTerminalValues(GammaValues(H, G))] = True

      MMn += [ X ]
      rrn += 1
      if rrn >= Buffer:
        M = Matrix(QQ, MMn)
        rrn = M.rank()
        P, L, U = M.LU()
        MMn = [ U[i].list() for i in range(rrn) ]
```

```
    Mn  = Matrix(Mn)
    MMn = Matrix(MMn)
    rn  = Mn.rank()
    rrn = MMn.rank()

    return [ n, rn, rrn ]
# Generate
```