Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Matchings tellen in kubische grafen

(Engelse titel: Counting matchings in cubic graphs)

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

PIM OTTE

Delft, Nederland
Juni 2014

BSc verslag TECHNISCHE WISKUNDE DRAFT

"Matchings tellen in kubische grafen"
(Engelse titel: "Counting matchings in cubic graphs")

PIM OTTE

Technische Universiteit Delft

Begeleider

dr. D. C. Gijswijt

Overige commissieleden

dr. ir. M.B. van Gijzen drs. E.M. van Elderen

Juni, 2014 Delft

Contents

Chapter 1

Introduction

In this bachelor thesis we will consider perfect matchings in a special type of graph. Recall that a graph is a structure consisting of *nodes* connected by *edges*. In traditional graphs each pair of nodes can only be connected by one edge. In this context we consider *multigraphs*, where multiple edges between a single pair of nodes are allowed. A *perfect matching* is a subset of the edges of a graph, such that each node is indicent to exactly one edge in this subset. For readability we will often use *matching*. Where we only consider partial matchings (where some nodes have no indicent edge), we will mention this explicitly. We will be counting the total number of matchings, so we will denote the set of all perfect matchings in a graph G by $\mathcal{M}(G)$. This means that we can write the total number of matchings as $|\mathcal{M}(G)|$

The special type of graph we are considering is *cubic*. This means that each node has exactly 3 edges connecting it to other nodes. Finally, we only consider connected *bridgeless* graphs. This entails that there is no edge we can remove such that the graph is divided into two components (i.e. two sets of vertices which are connected, but no path exists from one set to the other).

This is enough knowledge to introduce the Lovász-Plummer conjecture, which was formulated in [13, Conjecture 8.1.8].

Conjecture 1 (Lovász-Plummer)**.** There exists an $\epsilon > 0$ such that for any cubic bridgeless graph G the following holds:

$$2^{\epsilon|V(G)|} \leq |\mathcal{M}(G)|.$$

This conjecture was proven in 2011 by Louis Esperet, František Kardoš, Andrew D. King, Daniel Král and Serguei Norine [7]. In this proof they show that we may take $\epsilon = \frac{1}{3656}$. In addition, note that this conjecture does not specify that the graph has to be connected. However, if a graph is disconnected, we apply the proof to each component separately. Each matching of a disconnected component can be combined with all matchings of the other components to yield a matching for the original graph, which yields the result for the original graph. Therefore it is enough to consider connected graphs.

After definitions and background to this theorem in Chapter 1, we will provide examples and further explanation of the base concepts which are introduced or used in this proof in Chapters 2 and 3. In Chapter 4 we will provide a structural overview of this proof to enable further understanding. Chapter 5 will contain the main result of this thesis. In this chapter we will prove a stronger version of a corollary in the original proof. The consequences of this will be explored and this will result in an improvement of the constant to $\frac{1}{1686}$.

1.1 Definitions, notation and conventions

This thesis concerns graphs and (perfect) matchings. In this section we will state some basic definitions and notation that we will use throughout this thesis. We follow the notations and definitions from the paper of Esperet et al. [7]. We refer the reader to Diestel [4] for background and basic notation in graph theory.

For a graph G, we denote the set of vertices by $V(G)$ and the set of edges by $E(G)$. We will not consider graphs which have loops (edges from a node to itself), but we will consider graphs that have multiple edges between the same two nodes. For $X \subseteq V(G)$, we use $G|X$ for the subgraph of G induced by X. We define E_X as the set of all edges with at least one endpoint in X. The set $\delta(X)$ represents the set of edges with exactly one endpoint in X. This set is also called an *edge-cut* or *k-edge-cut* where $k = |\delta(X)|$, because these edges cut the graph into two *components* (or *sides*) when removed: $G|X$ and $G|(V(G) \setminus X)$. We call an edge-cut *cyclic* if both sides contain a cycle. If G contains no edge-cut of size less than k, G is *k-edge-connected.* Analogously, if G contains no cyclic edge-cut of size less than k, G is *cyclically k-edge-connected.*

Some graphs are used quite often, and are hence named. We use K_n to denote a graph on n nodes, each pair of which is connected by an edge. This is also known as the *complete* graph on n nodes. At some point we will also consider B_3. This is a graph on 2 nodes, connected by 3 parallel edges. Note that K_4 and B_3 are cubic graphs.

An important property of cuts in cubic graphs is that $(|\delta(X)| \equiv |X|) \mod 2$. We prove this by counting the number of indicent node-edge pairs. If $|X|$ is odd, the total number of node-edge pairs with the nodes in X is odd. There are an even number of node-edge pairs fully within X, hence $\delta(X)$ must be odd. If $|X|$ is even, the total number of node-edge pairs with nodes in X is even. There are an even number of these fully contained within X, so $\delta(X)$ is even. So in both cases we have the mentioned property.

In the introduction we defined $\mathcal{M}(G)$ as the set of perfect matchings of a graph G. The size of this set is the number of perfect matchings in this graph and this leads to the definition $m(G) = |\mathcal{M}(G)|$. In addition, we write $m^\star(G)$ for the minimum over all edges $e \in E(G)$ of the number of perfect matchings containing e.

Across this thesis we will come across some probability theory. We will adhere to the notation $\Pr[\mathbf{X} = x]$ for the probability of the event that $\mathbf{X}$ is equal to x and $\mathbb{E}[\mathbf{X}]$ for the expected value of $\mathbf{X}$. Finally, if we have a function $f : X \to Y$, we will refer to $f^{-1}(y) = \{x \in X : f(x) = y\}$ as the fiber of f at y.

1.2 Background to the theorem

The Lovász-Plummer conjecture has survived for a relatively long time before it was proven. Between the formulation and the resolution, progress had already been made with weaker forms of this conjecture. In this section we provide an overview of some of these related results, in order to give some context to this theorem.

The first type of earlier result considers additional restrictions on the type of graph. In 1979 Voorhoeve proved the conjecture for bipartite cubic graphs [15], with a lower bound of $\left(\frac{4}{3}\right)^{|V(G)|/2}$. In 1998 the bipartite case was extended further by Schrijver [14], who proved that all $k-$regular bipartite graphs have at least $\left(\frac{(k-1)^{k-1}}{k^{k-2}}\right)^{|V(G)|/2}$ perfect matchings. More recent work includes the version for a specific class of cubic graphs named fullerene graphs by Kardoš et al. [11], with a lower bound of $2^{\frac{|V(G)|-380}{61}}$ and a bound of $2^{|V(G)|/655978752}$ for cubic bridgeless planar graphs was proven by Chudnovsky and Seymour [2].

The second kind of earlier result does consider all cubic graphs, but instead provides a bound which is sub-exponential. Earlier results provide several types of linear bounds: $\frac{1}{4}|V(G)|+2$ was proven to be a lower bound by Edmonds, Lovász and Pulleybank [6]. Král et al. [12] improved this to $\frac{1}{2}|V(G)|$, followed by Esperet et al. [9] improving the bound to $\frac{3}{4}|V(G)| - 10$. Finally, Esperet, Kardoš and Král [8] showed that a superlinear bound exists.

This piece of mathematics has some interesting applications. Perfect matchings in cubic graphs are relevant for the Ising model for ferromagnetism [10]. The stability of fullerenes, a class of carbon molecules, relates directly to maximal matchings in cubic bridgeless graphs [1].

Chapter 2

Balanced probability distributions

In this chapter we will introduce balanced probability distributions, prove their existence in the case of cubic graphs and work out some small examples.

2.1 Definition

Given a graph G and $X \subseteq V(G)$, we define $\mathcal{M}(G,X)$ to be the family of subsets M of E_X, such that every vertex of X is incident with exactly one edge in M. Note that by defintion $\mathcal{M}(G,V(G)) = \mathcal{M}(G)$. Furthermore, elements of $\mathcal{M}(G,X)$ do not necessarily have to be partial matchings in G, because vertices outside X could be incident with multiple edges in this element.

Definition 1 (Balanced probability distribution)**.** A probability distribution $\mathbf{M}$ on $\mathcal{M}(G,X)$ is balanced if $\Pr[e \in \mathbf{M}] = \frac{1}{3}$ for every $e \in E_X$.

At first glance this may seem an arbitrary definition, but upon closer inspection this intuitively defines a "fair" distribution on $\mathcal{M}(G,X)$. Note that for each node all indicent edges have an equal probability of being in $\mathbf{M}$. Since the sum of the probabilities for all these edges should be one, and we will consider cubic graphs $\frac{1}{3}$ arises as a natural constant that all probabilities should be equal to.

2.2 Existence

Lemma 1 (Existence of a balanced distribution)**.** *Let G be a cubic bridgeless graph. Then there exists a balanced probability distribution on $\mathcal{M}(G)$.*

Proof. We use the characterization of the perfect matching polytope [5] to obtain a balanced distribution on $\mathcal{M}(G)$. Recall that a perfect matching can be described as a vector in $\{0,1\}^{E(G)}$, where an entry of the vector is 1 if and only if the corresponding edge is in the perfect matching. The perfect matching polytope describes all convex combinations of perfect matchings as a polytope in $\mathbb{R}^{E(G)}$. This characterization can be described as follows:

$$
\begin{aligned}
P_{matching}(G) = \{ \mathbf{x} \geq 0 :& \\
& x(\delta(v)) = 1 && \forall v \in V(G), \\
& x(\delta(U)) \geq 1 && \forall U \subseteq V(G) \text{ with } |U| \text{ odd} \\
\}.&
\end{aligned}
$$

In this characterization the convention is to write $x(\delta(U))$ for the sum over all variables that correspond to edges in $\delta(U)$. The non-negativity is a fairly straightforward constraint. The second constraint arises naturally from the fact that we consider perfect matchings, so the sum over all edges indicent to a node should be equal to one. The final constraint ensures that any point in this polytope is indeed the convex combination of perfect matchings. This constraint could be left out if we were to consider only bipartite graphs. We will prove that $\frac{1}{3}\mathbf{1}$ is an element of $P_{matching}(G)$ and that this induces the balanced probability distribution.

Firstly, we observe that all elements of this vector are larger than 0, so $\mathbf{x} \geq 0$ is satisfied. $x(\delta(v)) = 1$ is satisfied, because G is cubic. Let U be a subset of $V(G)$ with $|U|$ odd. Recall that $(|U| \equiv |\delta(U)|) \mod 2$, so $|\delta(U)|$ is odd. Because G has no bridge, the number of edges has to be strictly larger than 1. Hence $|\delta(U)| \geq 3$, so we have $x(\delta(U)) \geq 3 \cdot \frac{1}{3} = 1$.

This means that $\frac{1}{3}\mathbf{1}$ is a convex combination of perfect matchings. This induces a probability distribution by defining the chance of a matching M to be the coefficient in this convex combination. These coefficients are all between 0 and 1. Furthermore, the sum of all coefficients is 1 by definition of a convex combination. Therefore this convex combination gives a well-defined (though not necessarily unique) probability distribution. This probability distribution is balanced, because the sum of the probabilities over all matchings that contain an edge e is exactly $\frac{1}{3}$, which means it defines a balanced probability distribution. □

2.3 Properties

The first important property of balanced probability distributions is that they can be restricted. Let $\mathbf{M}_X$ be a balanced probability distribution on $\mathcal{M}(G, X)$ and let Y be a subset of X. Then we can define a balanced probability distribution $\mathbf{M}_Y$ as follows:

$$\Pr[\mathbf{M}_Y = M] := \sum_{M_X \in \mathcal{M}(G,X) : M \subseteq M_X} \Pr[\mathbf{M}_X = M_X] \qquad \forall M \in \mathcal{M}(G, Y)$$

Since each edge e in $\mathcal{M}(G, Y)$ is also an edge in $\mathcal{M}(G, X)$ we have that $\Pr[e \in \mathbf{M}_X] = \frac{1}{3}$. Since each element of $\mathcal{M}(G, X)$ can be restricted to $\mathcal{M}(G, Y)$, this probability is preserved and $\Pr[e \in \mathbf{M}_Y] = \frac{1}{3}$. Therefore, $\mathbf{M}_Y$ is a balanced probability distribution.

The distribution $\mathbf{M}_Y$ is referred to as the *restriction* of $\mathbf{M}_X$. The fact that balanced probability distributions can be restricted will allow us to define burls in Section 2.5 such that they have a similar property.

Secondly, we prove a lemma, which is Claim 3 in [7].

Lemma 2. *Let G be a cubic bridgeless graph and consider $Y \subseteq X \subseteq V(G)$ such that $C = \delta(Y)$ is a 3-edge-cut in G. For any balanced probability distribution $\mathbf{M}$ on $\mathcal{M}(G, X)$, and any $M \in \mathcal{M}(G, X)$ such that $\Pr[\mathbf{M} = M] > 0$, we have $|M \cap C| = 1$.*

Proof. Because $|\delta(Y)| = 3$, $|Y|$ must be odd. Hence, for any element M of $\mathcal{M}(G, X)$ it holds that $|M \cap \delta(Y)|$ is odd. A matching therefore intersects exactly 1 or all 3 of the edges in $\delta(Y)$. By definition of a balanced probaiblity distribution, the probability that $\mathbf{M}$ contains one specific edge is equal to $\frac{1}{3}$. There are four possibilities: Three cases where $\mathbf{M}$ intersects one specific edge and a fourth one where $\mathbf{M}$ intersects all the edges. We have estabilished that the first three cases have probability $\frac{1}{3}$, which sums to 1. Therefore, the probability of a matching containing all edges is 0. Hence if $\Pr[\mathbf{M} = M] > 0$, then $|M \cap C| = 1$ □

2.4 Example

Consider K_4, the complete graph on 4 nodes. This graph is cubic and bridgeless. We will denote the vertices by v_1, v_2, v_3, v_4 and the edge between v_i and v_j with e_{ij}. Note that $\mathcal{M}(K_4)$ contains 3 matchings: $\{e_{12}, e_{34}\}, \{e_{13}, e_{24}\}, \{e_{14}, e_{23}\}$. Because each edge is only in one matching, each of these three matchings needs to have probability $\frac{1}{3}$ in any balanced probability distribution. In this case the balanced probability distribution is unique, but this does not always have to be the case.

We will now consider balanced probability distributions on $\mathcal{M}(K_4, \{v_1, v_2\})$, which is equal to $\{\{e_{12}\}, \{e_{13}, e_{24}\}, \{e_{14}, e_{23}\}, \{e_{13}, e_{23}\}, \{e_{14}, e_{24}\}\}$. For any balanced probability distribution $\mathbf{M}$ it should hold that $\Pr[\mathbf{M} = \{e_{12}\}] = \frac{1}{3}$, because that is the only one to include e_{12}. Let $x = \Pr[\mathbf{M} = \{e_{13}, e_{24}\}]$. This yields $\Pr[\mathbf{M} = \{e_{13}, e_{23}\}] = \Pr[\mathbf{M} = \{e_{14}, e_{24}\}] = \frac{1}{3} - x$ and $\Pr[\mathbf{M} = \{e_{14}, e_{23}\} = x$. This gives a valid balanced probability distribution for $0 \leq x \leq \frac{1}{3}$. Note that for $x = \frac{1}{3}$ this gives the restriction of the balanced probability distribution on $\mathcal{M}(K_4)$.

2.5 Burls

Burls play a large part in the proof given in [7]. By $a(G, X, M)$ we denote the maximum number of disjoint M-alternating cycles in $G|X$.

Definition 2 (Burl)**.** A burl is a vertex set $X \subseteq V(G)$ such that for any balanced probability distribution $\mathbf{M}$ on $\mathcal{M}(G, X)$ it holds that $\mathbb{E}[a(G, X, \mathbf{M})] \geq \frac{1}{3}$.

Burls give a specific definition for the notion that a graph can have a lot of disjoint M-alternating cycles for some matching M. This notion is important, because a single perfect matching with N disjoint M-alternating cycles yields 2^N different perfect matchings and hence one can grasp that this will play an important role in the complete proof. Note that if $Y \subseteq X \subseteq V(G)$ and Y is a burl, then X is too. This conclusion can be drawn, because any balanced probability distribution on $\mathcal{M}(G, X)$ can be restricted to $\mathcal{M}(G, Y)$ and since $a(G, Y, M) \leq a(G, X, M)$ we get that $\mathbb{E}[a(G, X, \mathbf{M})] \geq \frac{1}{3}$, because Y is a burl.

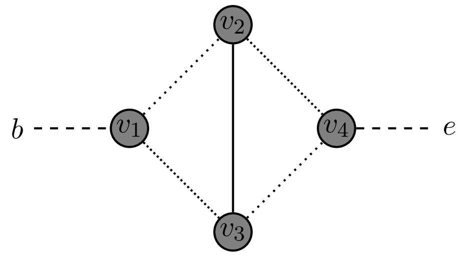


Figure 2.1: Small example of a burl

In Figure 2.1 we have provided a small example of a burl. If we consider a balanced probability distribution $\mathbf{M}$ on $\mathcal{M}(G, X)$ for $X = \{v_1, v_2, v_3, v_4\}$, then $\Pr[e_{b1} \in \mathbf{M}] = \frac{1}{3}$. Therefore $\Pr[e_{b1} \notin \mathbf{M}] = \frac{2}{3}$. The only elements of $\mathcal{M}(G, X)$ without e_{b1} are $\{e_{12}, e_{34}\}$ and $\{e_{13}, e_{24}\}$. These are marked by the dotted lines in the figure, and the union is an M-alternating cycle for either matching. Therefore with probability at least $\frac{2}{3}$ we have an M-alternating cycle, so X is a burl.

It turns out that if we have $X \subseteq V(G)$ with $\delta(X) \leq 4$ there are some simple sufficient criteria to conclude that X is a burl. We give the criteria here, however for the proof of these criteria we refer to [7] (Lemma 6 and 17).

Lemma 3. *Any $X \subseteq V(G)$ is a burl if it satisfies one of the following conditions*

- $\delta(X) = 2$
- $\delta(X) = 3$ *and* $|X| \geq 5$
- $\delta(X) = 4$ *and* $m(G|X) \geq 2$.

Chapter 3

Cut contractions and decompositions

In this chapter we will introduce cut contractions and cut decompositions. Intuitively, cut decompositions are a method to decompose a graph into smaller graphs which are connected by cuts. Cut contractions give us a way to reason about the original graph given a cut decomposition.

3.1 Cut contractions

Definition 3 (Cut contraction)**.** Let C be a 2- or 3-edge-cut in a graph G. Then the two C-contractions are defined as follows:

- If $C = \{e_1, e_2\}$, we take a side of C and remove e_1, e_2, as well as the other side. We then connect the two nodes from which e_1 and e_2 were removed.
- If $|C| = 3$, we take a side of C and identify all the vertices on the other side with each other.

Depending on which side we take, we get two different $C-$contractions. We say the resulting graphs G_1, G_2 are obtained from G by a cut-contraction. In the case of the 3-edge-cut, we turn all the nodes of one side X into one node, and of edges in E_X we only preserve $\delta(X)$. Therefore, in the case of a 3-edge-cut, we add a node, which is referred to as the *new node*. In the case of a 2-edge-cut we add only an edge. This edge is referred to as the *new edge*.

Note that G_1, G_2 have a couple of nice properties, among which: If G is cubic and bridgeless, then so are G_1 and G_2, and $m^\star(G) \geq m^\star(G_1)m^\star(G_2)$. For a proof of these properties, we refer to [7].

3.2 Cut decompositions

Definition 4 (Cut decomposition)**.** Let G be a graph. A pair (T, ϕ) is a non-trivial cut-decomposition if both of the following properties are satisfied:

- T is a tree with $E(T) \neq \emptyset$ and
- $\phi : V(G) \to V(T)$ is a function, with $|\phi^{-1}(t)| + \deg_T(t) \geq 3$ for each $t \in V(T)$.

Again this definition seems fairly arbitrary, and at first glance it may not be clear even why this is called a cut-decomposition. This name has been chosen because each edge in T corresponds to a cut in G. Given a graph G and a non-trivial cut-decomposition we can induce a cut in G for every edge $f \in T$ as follows. Let T_1, T_2 be the components of $T \setminus f$. Then

$(X_1, X_2) = (\phi^{-1}(T_1), \phi^{-1}(T_2))$ is an ordered partition of $V(G)$ and we denote the cut $\delta(X_1) = \delta(X_2)$ by $\phi^{-1}(f)$.

Definition 5 (Small-cut-decomposition)**.** If $|\phi^{-1}(f)| \in \{2, 3\}$ for each $f \in E(T)$, then we call (T, ϕ) a small-cut-decomposition of G.

3.2.1 Example of a small-cut-decompostion

An example of a small-cut-decomposition is sketched in Figure 3.1. The left graph is a cubic bridgeless graph, and the right graph is the tree to which it is mapped. The function ϕ is represented by the numbers in the nodes, if a node is marked with a 1 in the graph, ϕ maps it to the node marked 1 in the tree. By inspection one can see that this is not the only possible small-cut-decomposition for the given graph. To reason about small-cut-decompositions we will need more information. One method of providing this is to consider hubs.

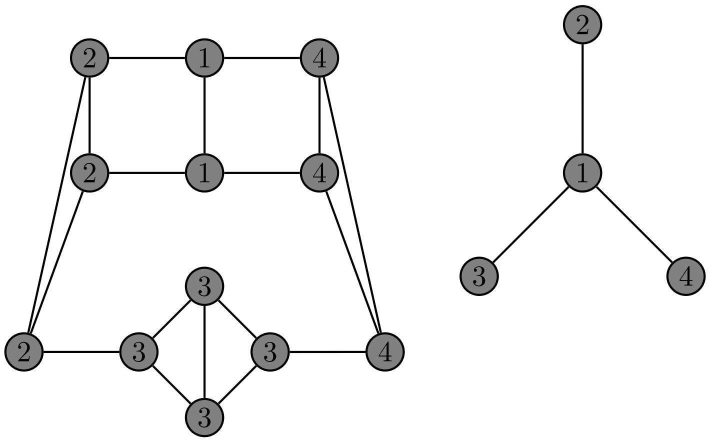


Figure 3.1: Example of a small-cut-decomposition

Let (T, ϕ) be a small cut decomposition of a bridgeless cubic graph G, and let T_0 be a subtree of T, such that $\phi^{-1}(T_0) \neq \emptyset$. Let $T_1, \ldots, T_n$ be the components of $T \setminus V(T_0)$, and for $0 \leq i \leq n$, let f_i be the unique edge of T with an end in $V(T_0)$ and $V(T_i)$. In addition, we write $X_i = \phi^{-1}(V(T_i))$.

Definition 6 (Hub)**.** Let G' be the graph obtained from G as follows. Set $G_0 = G$. For $1 \leq i \leq n$, take G_{i-1} and let G_i be the $(\phi^{-1}(f_i))-$contraction containing X_0. We define $G' = G_n$. We call G' the hub of G at T_0.

3.2.2 Example of a hub

In Figure 3.2 we provide an example of a hub. We take T_0 to be the node marked 1. To get this hub, three cut contractions have been made. One 2-cut contraction and two 3-cut contractions. The nodes marked $2'$ and $4'$ are the new nodes created in the 3-cut contractions and the edge between these nodes is the one that was created in the 2-cut contraction. Note that this graph is cubic and that $\phi^{-1}(T_0)$ and all outgoing edges are the same as they were in G.

3.2.3 Existence of small-cut-decompositions

Let $\mathcal{Y}$ be a collection of disjoint subsets of $V(G)$. We say that a small-cut-decomposition (T, ϕ) of G *refines* $\mathcal{Y}$ if for every $Y \in \mathcal{Y}$ there exists a leaf $v \in V(T)$ such that $Y = \phi^{-1}(v)$.

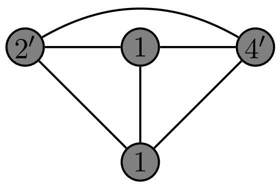


Figure 3.2: Hub at the red node from Figure 3.1

The following Lemma is given in [7] as Lemma 14 with partial proof. Here we provide all cases.

Lemma 4. *Let G be a cubic bridgeless graph. Let $\mathcal{Y}$ be a collection of disjoint subsets of $V(G)$ such that $|Y| \geq 2$ and $|\delta(Y)| \in \{2,3\}$ for every $Y \in \mathcal{Y}$. If one of the following conditions holds, there exists a small-cut-decomposition refining $\mathcal{Y}$*

1. *$\mathcal{Y} = \emptyset$ and G is not cyclically 4-edge-connected.*
2. *$\mathcal{Y} = \{Y\}$ and $|V(G) \setminus Y| > 1$*
3. *$\mathcal{Y} \geq 2$.*

Proof. Let $\mathcal{Y} = \emptyset$ and G bridgeless, cubic and not cyclically 4-edge-connected. Take $X \subset V(G)$ such that $\delta(X)$ is a cyclic-edge cut with $|\delta(X)| \leq 3$. Define T the tree with two nodes, t, t' with one edge. We define $\phi(x) = t$ if $x \in X$ and $\phi(x) = t'$ otherwise. This is a cut-decomposition, because $|\phi^{-1}(s)| \geq 2$ for $s \in \{t, t'\}$, because there are cycles in the components. This is a small-cut-decomposition, because $\phi^{-1}(e) = \delta(X)$, which is of size 2 or 3.

Let $\mathcal{Y} = \{Y\}$ and $|V(G) \setminus Y| > 1$. Take T the same tree as above and define $\phi(x) = t$ if $x \in Y$ and $\phi(x) = t'$ otherwise. Again $|\phi^{-1}(s)| \geq 2$ for $s \in \{t, t'\}$. By definition of Y this is also a small-cut-decomposition.

As third case we consider $\mathcal{Y} = \{Y_1, Y_2\}$. First we consider the case in which $V(G) = Y_1 \cup Y_2$ For this the exact same small-cut-decomposition as in the case $|\mathcal{Y}| = 1$, applied to $Y = Y_1$, is also a small cut decomposition for this case, and it also refines Y_2. If there are nodes which are neither in Y_1, nor Y_2, the same construction as we will give for $|\mathcal{Y}| \geq 3$ is sufficient.

If $|\mathcal{Y}| \geq 3$ we take T the tree on $|\mathcal{Y}| + 1$ vertices with $|\mathcal{Y}|$ leaves, which we denote by v_Y for each $Y \in \mathcal{Y}$ and we denote the central vertex by v_0. We define ϕ by $\phi(u) = v_Y$ if $u \in Y$ for some $Y \in \mathcal{Y}$ and $\phi(u) = v_0$ otherwise. The pair (T, ϕ) by definition refines $\mathcal{Y}$, the pair clearly satisfies all conditions to be a cut-decomposition and each edge corresponds to a cut induced by the pre-image of a leaf and hence is of size 2 or 3 .

□

Chapter 4

Structure of proof of Esperet et al.

To provide some insight in the proof given in [7] we provide a dependency graph of all claims, lemmas, corollaries and theorems that appear in the arXiv version of this paper. We will prove a stronger version of Corollary 21, without making use of Lemma 18, Lemma 19 or Corollary 20. Our proof does lean on Lemma 17, but only in a small way.

We will briefly describe the function of each cluster visible in this dependency graph. Lemmas 8, 9 and 10 eliminate certain triangles. Lemmas 14 and 15 and Corollary 16 prove the existence of small-cut-decompositions refining certain families. Lemma 7 states some basic properties of cut-contractions. Claim 3 and Lemmas 5 and 6 show that (large enough) subsets X with $|\delta(X)| \in \{2, 3\}$ are burls. Corollary 21 and all supporting statements combine small-cut-decompositions with burls to show how a long path in a small-cut-decomposition results in a burl. Lemmas 22 and 23 provide some facts about cyclically 4-edge-connected graphs and Lemmas 11 and 13 with Corollary 12 support the proof of Theorem 2, which is a reformulation of Theorem 1.

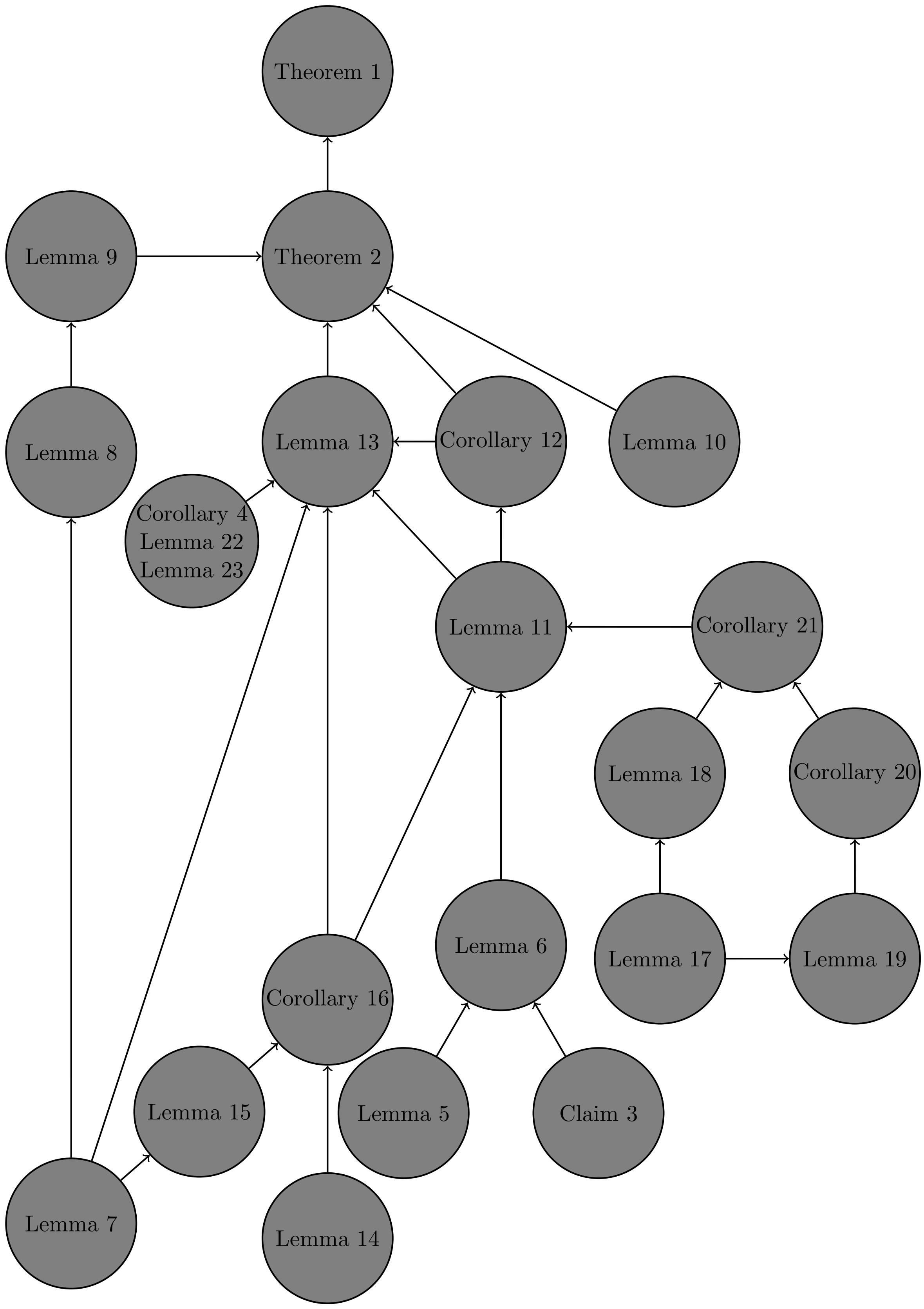


Figure 4.1: Dependency graph of all statements in the proof

Chapter 5

Improving the bound

As the authors suggest in Section 6 of [7], the constant in the main theorem can be improved. In particular, it is Corollary 21 that can be improved. This corollary describes that a long path in a small-cut-decompositions leads to a burl. In this chapter we will prove the main theorem (Theorem 1) of this thesis and demonstrate how this improves the bound.

Theorem 1 (Long paths in small-cut-decompositions)**.** *Let (T, ϕ) be a small-cut-decomposition of a cubic bridgeless graph G and let P be a path in T with $|V(P)| = 11$. If for every $t \in V(P)$, $\deg_T(t) = 2$ and the hub of G at t is isomorphic to K_4 or B_3, then $\phi^{-1}(P)$ is a burl.*

Recall that K_4 is the complete graph on 4 nodes and B_3 is the graph consisting of 2 nodes connected with 3 edges.

We will prove this theorem by characterizing $\phi^{-1}(t)$ for all $t \in V(P)$ and considering how each of the subgraphs $G|(\phi^{-1}(t))$ allows continuation of a perfect matching. By case analysis we supply sufficient conditions under which $V(P)$ is a burl. Finally, we supply an algorithm to check all possible paths constructed from the characterizations.

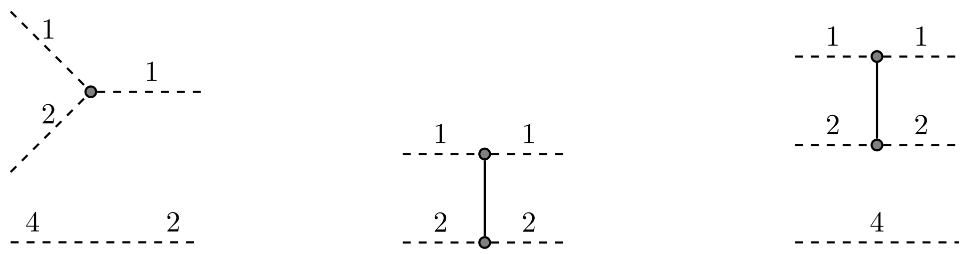
5.1 Characterizing the path

Consider a node t of degree 2 in a small-cut-decomposition. The general strategy is to describe how $\phi^{-1}(t)$ depends on the hub at t. We know this hub is isomorphic to either B_3 or K_4. Because $\deg_T(t) = 2$, these hubs are the result of exactly two cut-contractions which notably preserve $\phi^{-1}(t)$. Recall that a 3-cut contraction adds a new node, whereas a 2-cut contraction does not. This means we can determine how many of each type we have when we know $|\phi^{-1}(t)|$. In addition, note that $|\phi^{-1}(t)| \geq 1$ by the definition of a small-cut-decomposition. We will consider two cases.

5.1.1 The hub at t is isomorphic to B_3

Since B_3 contains only two nodes, $|\phi^{-1}(t)| \leq 2$. If $|\phi^{-1}(t)| = 1$, the cuts must be a 2-cut and a 3-cut contraction. The resulting situation is sketched in Figure 5.1a (The numbers in this and all following figures will be relevant when we view the subgraphs in these figures as functions in a later section.)

The other case is $|\phi^{-1}(t)| = 2$. In this case the cuts are both 2-cuts, since no new nodes can be created by cuts. These two nodes must be connected by an edge, otherwise there could not be the 3 edges needed for B_3. Hence, in this case $\phi^{-1}(t)$ is as depicted in in Figure 5.1b.



(a) B_3 and $|\phi^{-1}(t)| = 1$ (b) B_3 and $|\phi^{-1}(t)| = 2$ (c) K_4 and $|\phi^{-1}(t)| = 2$

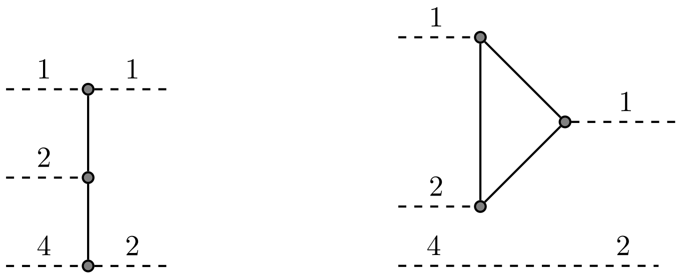
Figure 5.1: $\phi^{-1}(t)$ for several hubs and sizes

5.1.2 The hub at t is isomorphic to K_4

In this case we can deduce that $2 \leq |\phi^{-1}(t)| \leq 4$, because the two cut-contractions both add none or one node. Again, we separate on these sizes:

If $|\phi^{-1}(t)| = 2$, both contractions must be 3-cut contractions. Once again, the two nodes must be connected, because in 3-cut contractions no new edges are created. This gives us the situation as in Figure 5.1c

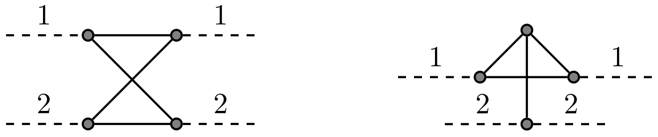
If $|\phi^{-1}(t)| = 3$, one contraction must be a 3-cut contraction and the other must be a 2-cut contraction. Because the 2-cut contraction induces an extra edge, there can be one missing edge within $\phi^{-1}(t)$. The case with one missing edge is demonstrated in Figure 5.2a, the case without missing edges can be found in Figure 5.2b. Note that the second possibility is not relevant for the purposes of proving Theorem 1, since we can redefine any small-cut-decomposition including this case to split into in the two cases highlighted in Figures 5.1a and 5.1b.



(a) K_4 and $|\phi^{-1}(t)| = 3$ (b) K_4 and $|\phi^{-1}(t)| = 3$

Figure 5.2: $\phi^{-1}(t)$ for several hubs and sizes

The last case is $|\phi^{-1}(t)| = 4$. This cases induces two 2-cut contractions. This means that $\phi^{-1}(t)$ is isomorphic to K_4 with two edges removed. Again this yields two possibilities:



(a) K_4 and $|\phi^{-1}(t)| = 4$ (b) K_4 and $|\phi^{-1}(t)| = 4$

Figure 5.3: $\phi^{-1}(t)$ for several hubs and sizes

In this case 5.3a can be left out without loss of generality. Since it contains a 4-cycle, any path for which there is a $\phi^{-1}(t)$ of this type, is automatically a burl, by Lemma 3.

This concludes all possiblilities for $\phi^{-1}(t)$ we can encounter in proving Theorem 1.

5.2 Converting the fibers to functions

From now on, we will refer to $\phi^{-1}(t)$ as a *fiber*, if it does not concern a specific node t. We will refer to $\phi^{-1}(V(P))$ as the *path fiber*. If we refer to a *node* without specifying, this is a node $t \in P$.

By the nature of the small-cut-decomposition, we know each fiber $\phi^{-1}(t)$ is seperated from the rest of the graph by two cuts. If we consider the path in one direction, for each fiber this defines two cuts: an incoming cut (C_{in}), and an outgoing cut (C_{out}). This allows us to view each element as a function $\psi : \mathcal{P}(C_{in}) \times \mathcal{P}(C_{out}) \to \mathbb{N}$. defined as:

$$\psi(F, F') = |\{M \in \mathcal{M}(G, \phi^{-1}(t)) : M \cap C_{in} = F \text{ and } M \cap C_{out} = F'\}|$$

This function represents how many elements exist in $\mathcal{M}(G, X)$ containing specific subsets of C_{in} and C_{out}, where X is a fiber. We can also represent this with a matrix A_t defined by $(A_t)_{F,F'} = \phi(F, F')$. This is a matrix of size $2^{|C_{in}|} \times 2^{|C_{out}|}$. In Figures 5.4 and 5.5 we provide all the matrices for the relevant fibers. For the purposes of display and later calculations we use the following construction to determine the order: We associate each element S of $\mathcal{P}(C_{in})$ with an integer from 0 to $2^{|C_{in}|} - 1$ in the following manner. We add all the numbers from the figures in Section 5.1 of each edge in S. So $S = \emptyset$ corresponds to 0 and $S = C_{in}$ corresponds to 3 in the case of a 2-edge cut and 7 in the case of a 3-edge cut. This encoding determines the order in the matrices below and will also return later.

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(a) Matrix for Figure 5.1a

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

(b) Matrix for Figure 5.1b

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(c) Matrix for Figure 5.1c

Figure 5.4: Matrices corresponding to several elements.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) Matrix for Figure 5.2a

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

(b) Matrix for Figure 5.3b

Figure 5.5: Matrices corresponding to several elements.

5.3 Representing a path fiber by a matrix

If we have a path $P = (t_1, t_2, \ldots, t_n)$ in the tree T of a small-cut-decomposition, we can apply the same technique and represent the path fiber as a function which gives the number of elements in $\mathcal{M}(G, X)$ having the prescribed intersection with the cuts C_{in} and C_{out}, with $X = \phi^{-1}(P)$. Furthermore, The matrix A_P corresponding to this function is exactly the product of the matrices representing each fiber: $A_P = A_{t_n} \cdot A_{t_{n-1}} \cdots A_{t_2} \cdot A_{t_1}$.

Given a matrix of a path, the following method suffices to prove that the path fiber is a burl. We find a subset S of $\mathcal{P}(C_{in})$ such that the probability of an element occurring is at least $\frac{1}{3}$. If the number of matchings is unequal to 1 for each element of this subset and each subset of C_{out}, then the path fiber is a burl. The reason is as follows. For each element of $\mathcal{M}(G, X)$ containing one of the elements in S, there is another matching with exactly the same edges in C_{out}. Hence, there is an M-alternating cycle in $G|X$. Because this situation occurs with probability at least $\frac{1}{3}$, this yields $\mathbb{E}(a(G, X, \mathbf{M})) \geq \frac{1}{3}$. Similarly, we can find a subset of $\mathcal{P}(C_{out})$ such that the number of matchings is unequal to 1 for each element of this subset and each subset of C_{in}.

Note that for purposes of determining whether a matrix represents a burl or not, we can at any point in the multiplication that gives A_P take all numbers larger than two in the matrix and redefine them to be two. We only check which numbers are not equal to one, and if another number is multiplied by the number which we decreased, it will still be larger. While this seems a fairly innocuous operation, this allows us to consolidate a lot of paths which seem different, but have the same main characteristics relevant to burls.

5.4 Matrices representing burls

We want to determine a subset S of $\mathcal{P}(C_{in})$ such that the probability of S in any balanced probability distribution is larger than or equal to $\frac{1}{3}$. We define S_i as the subset of $\mathcal{P}(C_{in})$ encoded in binary by i as in Section 5.3. Let $\mathbf{M}$ be a balanced probability distribution on $\mathcal{M}(G, X)$. Then we say that $p_i = \Pr[\mathbf{M} \cap C_{in} = S_i]$. We consider a set of equations that p_i should satisfy for any balanced probability distribution.

For $|C_{in}| = 2$, these probabilities satisfy the following equations. The last two follow from the fact that we are dealing with a balanced probability distribution.

$$\begin{array}{ccccccccc} p_0 & + & p_1 & + & p_2 & + & p_3 & = & 1 \\ & & p_1 & & & + & p_3 & = & \frac{1}{3} \\ & & & & p_2 & + & p_3 & = & \frac{1}{3} \end{array}$$

This yields $p_0 \geq \frac{1}{3}$ (by substraction of the second and third equation from the first). Hence, a matrix representing a path fiber with 4 columns represents a burl if one of the following sets of columns contains no ones: $\{0\}, \{1, 3\}, \{2, 3\}$.

For $|C_{in}| = 3$, these probabilities satisfy the following equations. Here, the last three correspond to the balance of the probability distribution.

$$\begin{array}{ccccccccccccccccc} p_0 & + & p_1 & + & p_2 & + & p_3 & + & p_4 & + & p_5 & + & p_6 & + & p_7 & = & 1 \\ & & p_1 & & & + & p_3 & & & + & p_5 & & & + & p_7 & = & \frac{1}{3} \\ & & & & p_2 & + & p_3 & & & & & + & p_6 & + & p_7 & = & \frac{1}{3} \\ & & & & & & & & p_4 & + & p_5 & + & p_6 & + & p_7 & = & \frac{1}{3} \end{array}$$

Taking the first equation and subtracting the third and fourth yields that $p_0 + p_1 \geq \frac{1}{3}$, so by symmetry and the original equations this yields that the following combinations of indices for columns give enough information to conclude the path fiber represented by this matrix is a burl

$$\{0, 1\}, \{0, 2\}, \{0, 4\}, \{1, 3, 5, 7\}, \{2, 3, 6, 7\}, \{4, 5, 6, 7\}.$$

Hence if the columns with these indices contain no ones, the matrix represents a burl.

Of course the exact same reasoning holds for C_{out} and the rows of the matrix.

These two reasonings constitute a basic way to determine if a matrix represents a burl. The final method yields stronger results, but takes more computing time to check.

Let A_P be a matrix representing a path P, and X the path fiber. We define $\tau : \mathcal{M}(G, X) \to \mathcal{P}(C_{in}) \times \mathcal{P}(C_{out})$ as

$$\tau(M) = (M \cap C_{in}, M \cap C_{out})$$

We want to show that $\Pr[\psi(\tau(\mathbf{M})) = 1] \leq 2 \Pr[\psi(\tau(\mathbf{M})) \neq 1]$. In words: we want the probability that an element of $\mathcal{M}(G, X)$ is the only matching with a certain incoming cut and outcoming cut to be bounded by two times the probability that this is not the case. In this case we can conclude that the probability that $\mathbf{M}$ has an alternating circle is at least $\frac{1}{3}$.

Let p_{ij} be the probability that an element of a balanced probability distribution corresponding to incoming cut encoding j and outcoming cut encoding i. We define

$$c_{ij} = \begin{cases} 2 & \text{if } (A_P)_{ij} \neq 1 \\ -1 & \text{if } (A_P)_{ij} = 1 \end{cases}$$

We consider a linear program with the following objective:

$$\min : \sum_{i,j} c_{ij} p_{ij}$$

The first condition should be $\sum_{i,j} p_{ij} = 1$. Furthermore, the sums of probabilities over a row or column should satisfy the same conditions as in the base case. We use the variables r_i, c_i for rows and columns respectively. We formulate the full linear program for a 4×4 matrix. The other cases are analogous.

$$\begin{aligned} \text{minimize} \quad & \sum_{0 \leq i,j \leq 3} c_{ij} p_{ij} \\ \text{subject to} \quad & \sum_{0 \leq i,j \leq 3} p_{ij} = 1 \\ & p_{ij} \geq 0, \qquad \forall i, j \text{ with } 0 \leq i, j \leq 3 \\ & \sum_{0 \leq i \leq 3} p_{ij} = c_j, \ \forall j \text{ with } 0 \leq j \leq 3 \\ & \sum_{0 \leq j \leq 3} p_{ij} = r_i, \ \forall i \text{ with } 0 \leq i \leq 3 \\ & \sum_{i \in F} r_i = \frac{1}{3}, \qquad \forall F \in \{\{1,3\},\{2,3\}\} \\ & \sum_{i \in F} c_i = \frac{1}{3}, \qquad \forall F \in \{\{1,3\},\{2,3\}\} \end{aligned}$$

If this linear program yields a value larger than or equal to 0, we can conclude that the matrix represents a burl. The reasoning behind this is that all balanced probability distributions satisfy the above equations, so if our target function is positive for all of these possibilities, it must also be for all balanced probability distributions and that is what we wanted to show.

5.5 Using the characterization to check all paths

We have now characterized all fibers in a path in the small-cut-decomposition. In addition, we have shown that each of these fibers, and the path fiber can be represented by a matrix. Finally, we have discerned sufficient conditions for a matrix such that we can conclude that the sequence of elements represented by this matrix is a burl. Note that if a path has a subpath and the path fiber of this subpath represents a burl, then the path is a burl. This is one of the basic properties of a burl mentioned in Section 2.5.

We can proceed in the following manner. For path lengths from 1 to 11 we take the following approach: For length 1 we consider all matrices that represent a fiber, for length $1 < n \leq 11$ we consider all matrices that represent non-burl path fibers, where the path is of length $n-1$. Note that throughout this algorithm we allow no numbers larger than 2 as described earlier. We multiply all these matrices with all possible permutation matrices followed with the multiplication by all possible matrices representing fibers which were described in Section 5.2. We multiply the matrices which do not represent a burl from both the left and the right. We will only check a matrix if it is a result of both a multiplication from the left and a multiplication from the right. If it is only the result of one of these, then the path of length $n-1$ which was not used in the construction did correspond to a burl, and hence the path fiber for the path of length n corresponds to a burl as well. Then we check if the resulting matrix represents a burl. If not, we save it for the next iteration (again, with no numbers larger than 2).

Finally, we keep track of the number of 2-edge-cuts in the graph. This is because any path with three 2-edge-cuts in it is a burl by Lemma 17 of [7].

Theorem 2 (Matrices representing burls)**.** *Let (T, ϕ) be a small-cut-decomposition of a cubic graph G. Let P be a path of length 11 such that for each node t in P it holds that $\deg_T(t) = 2$ and the hub at t is isomorphic to B_3 or K_4. Then P has one of the following properties:*

- *The matrix representing the path fiber represents a burl according to one of the criteria in Section 5.4.*
- *P has a subpath for which the matrix representing the path fiber represents a burl according to the same criteria.*
- *P has 3 or more edges f such that $|\phi^{-1}(f)| = 2$*

Proof. To implement the algorithm described in the above section, we used Python 2.7[1]. To solve the linear progams needed, we called lp_solve[2] from Python. We provide the final code in Appendix A

Running this code this, it turns out that this process converges and after considering paths of length 11, we have no non-burls left. □

5.6 Calculating the consequences

Theorem 2 immediately results in Theorem 1. We have shown that we only need fairly short paths to obtain burls. When we consider the implications for the proof, we consider the system of inequalities in [7], Section 2.1. The original system is as follows:

[1] `https://www.python.org/download/releases/2.7/`

[2] `http://lpsolve.sourceforge.net/5.5/`

$$0 < \alpha \leq \beta_2 \leq \beta_1 \tag{5.1}$$
$$1/3656 \leq \frac{\alpha}{9\beta_1 + 3} \tag{5.2}$$
$$\beta_2 + 6\alpha \leq \beta_1 \tag{5.3}$$
$$74\alpha \leq \beta_2 \tag{5.4}$$
$$146\alpha \leq \beta_1 \tag{5.5}$$
$$\beta_2 + 80\alpha \leq \beta_1 \tag{5.6}$$
$$6\alpha + \gamma \leq \log(6)/\log(2) \tag{5.7}$$
$$\gamma + 2\beta_1 + 7\alpha - \beta_2 \leq 1 \tag{5.8}$$
$$6\alpha + 2\beta_1 \leq \log(\frac{4}{3})/\log(2) \tag{5.9}$$
$$2\beta_1 + 4\alpha \leq \gamma \tag{5.10}$$

We can inspect the proof and express these constraints in terms of the length l. In the original proof this was 32, so substituting $l = 32$ results in the above equations. We substitute $l = 11$ to get the new inequalities. This replaces inequalities 4, 5 and 6 with:

$$(10 + 2l)\alpha = 32\alpha \leq \beta_2 \tag{5.4}$$
$$(18 + 4l)\alpha = 62\alpha \leq \beta_1 \tag{5.5}$$
$$\beta_2 + (16 + 2l)\alpha = \beta_2 + 38\alpha \leq \beta_1 \tag{5.6}$$

Using lp_solve, we can determine that with these new inequalities, the 3656 in inequatility 2 can be reduced to 1686 and this system still has a solution. For 1685 this problem becomes infeasible.

In the new solution equations (5.4), (5.6), (5.9) (5.10) are tight with the following solution being feasible:

$$\alpha = \frac{\log(4/3)}{146\log(2)}$$
$$\beta_1 = \frac{70\log(4/3)}{146\log(2)}$$
$$\beta_2 = \frac{32\log(4/3)}{146\log(2)}$$
$$\gamma = \frac{144\log(4/3)}{146\log(2)}$$

Chapter 6

Concluding remarks

In this thesis we have provided background and explanation to a fairly recent piece of mathematics. By means of examples and small proofs the core elements of the proof of Esperet et al. [7] have been dissected. We have shown that this proof can be strengthened by carefully considering all relevant cases with an algorithm. This results in an improved constant in the proof of the Lovász-Plummer conjecture.

Though we do not provide proof of this, we strongly suspect that the path length in Theorem 1 cannot be reduced to less than 10. Partially this is because Lemma 18 in [7], which concerns a more limited case is already at length 10. This means that this part of the proof might be slightly more improved, but any other advancements should be in other parts of the proof, or using an entirely new technique altogether.

Other branches of relevant research include finding an upper bound on the number of perfect matchings. Cygan et al. [3] have shown that for some constant c there exists a cubic bridgeless graph with at least n vertices and at most $c \cdot 2^{n/17.285}$ perfect matchings.

A more general, unsolved conjecture is formulated in [13, Conjecture 8.1.8].

Conjecture 2. For $k \geq 3$ there exist constants $c_1(k), c_2(k) > 0$, such that every $k-$regular matching-covered graph contains at least $c_2(k)c_1(k)^{|V(G)}$ perfect matchings. Furthermore, $c_1(k) \to \infty$ as $k \to \infty$.

In this conjecture, the term matching-covered is used. A graph is *matching-covered* if every edge of the graph belongs to a perfect matching. It appears that the proof of Esperet et al. does not generalize to this conjecture.

6.1 Acknowledgements

There are some people who have contributed to this thesis, directly or indirectly and I would like to take this opportunity to express my gratitude. First of all, dr. Dion Gijswijt for his encouragement during the whole process and particularly his suggestion of this topic, which I have enjoyed tremendously. I thank prof. dr. Karen Aardal for her introduction to optimization as a whole and her patience with me during my honours programme research. I would like to thank Arthur Bik and Laura Molenaar for the useful feedback that resulted from their proofreading of this thesis. Finally, I want to thank Lidwien, Bernard and Arien, who have all inspired and encouraged me in their own way.

Bibliography

[1] Vesna Andova, František Kardoš, and Riste Škrekovski. Sandwiching saturation number of fullerene graphs. *arXiv preprint arXiv:1405.2197*, 2014.

[2] Maria Chudnovsky and Paul Seymour. Perfect matchings in planar cubic graphs. *Combinatorica*, 32(4):403–424, 2012.

[3] Marek Cygan, Marcin Pilipczuk, and Riste Škrekovski. A bound on the number of perfect matchings in klee-graphs. *Discrete Mathematics & Theoretical Computer Science*, 15(1):37–54, 2013.

[4] Reinhard Diestel. *Graph theory. 2005.* Springer, 2010.

[5] Jack Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *J. Res. Nat. Bur. Standards B*, 69:125–130, 1965.

[6] Jack Edmonds, WR Pulleyblank, and L Lovász. Brick decompositions and the matching rank of graphs. *Combinatorica*, 2(3):247–274, 1982.

[7] Louis Esperet, František Kardoš, Andrew D King, Daniel Král, and Serguei Norine. Exponentially many perfect matchings in cubic graphs. *Advances in Mathematics*, 227(4):1646–1664, 2011.

[8] Louis Esperet, František Kardoš, and Daniel Král. A superlinear bound on the number of perfect matchings in cubic bridgeless graphs. *European Journal of Combinatorics*, 33(5):767–798, 2012.

[9] Louis Esperet, Daniel Kral, Petr Škoda, and Riste Škrekovski. An improved linear bound on the number of perfect matchings in cubic graphs. *European Journal of Combinatorics*, 31(5):1316–1334, 2010.

[10] Andrea Jiménez and Marcos Kiwi. Antiferromagnetic ising model in triangulations with applications to counting perfect matchings. *Discrete Applied Mathematics*, 172:45–61, 2014.

[11] František Kardoš, Daniel Král, Jozef Miškuf, and Jean-Sébastien Sereni. Fullerene graphs have exponentially many perfect matchings. *Journal of mathematical chemistry*, 46(2):443–447, 2009.

[12] Daniel Král', Jean-Sébastien Sereni, and Michael Stiebitz. A new lower bound on the number of perfect matchings in cubic graphs. *SIAM Journal on Discrete Mathematics*, 23(3):1465–1483, 2009.

[13] Michael D Plummer and László Lovász. *Matching theory.* Elsevier, 1986.

[14] Alexander Schrijver. Counting 1-factors in regular bipartite graphs. *Journal of Combinatorial Theory, Series B*, 72(1):122–135, 1998.

[15] Marc Voorhoeve. A lower bound for the permanents of certain (0, 1)-matrices. In *Indagationes Mathematicae (Proceedings)*, volume 82, pages 83–86. Elsevier, 1979.

Appendix A

Code

```python
#!/usr/bin/python2

import numpy as np
import itertools
import sys
import string
import json
from subprocess import check_output



def generate_permutation_matrices(n):
    """Generates all matrices representing permutations.
    """
    #Generate a permutation of (0, 1, ... n)
    for p in itertools.permutations(range(n)):
        mat = np.zeros((2**n, 2**n))
        #Determine how encoding i is permuted
        for i in xrange(2**n):
            binary = format(i,"0" + str(n) + "b")
            permutedbinary = ""
            for pos in p:
                permutedbinary += binary[pos]
            mat[i,int(permutedbinary,2)] = 1
        yield np.asmatrix(mat, dtype=np.int32)

perm_matrices = {}
perm_matrices[4] = [mat for mat in generate_permutation_matrices(2)]
perm_matrices[8] = [mat for mat in generate_permutation_matrices(3)]

#Vertex matrices as described in Section 5.2
vertex_matrices = {}
m1 = np.matrix([[1, 0, 0, 1, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0],
                [0, 1, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 1, 0, 0, 1],
                [0, 0, 0, 0, 0, 0, 1, 0],
                [0, 0, 0, 0, 0, 1, 0, 0],
                [0, 0, 0, 0, 1, 0, 0, 0]
               ], dtype=np.int32)

m2 = np.matrix([[0, 1, 0, 0, 1, 0, 0, 1],
                [1, 0, 0, 0, 0, 0, 1, 0],
                 [1, 0, 0, 1, 0, 0, 0, 0],
                 [0, 0, 1, 0, 0, 0, 0, 0]
                ], dtype=np.int32)

m3 = np.matrix([[0, 1, 1, 0, 0, 0, 0, 0],
                 [1, 0, 0, 0, 0, 0, 0, 0],
                 [0, 0, 0, 0, 0, 1, 1, 0],
                 [0, 0, 0, 0, 1, 0, 0, 0]
                ], dtype=np.int32)

m4 = m1[:4,:4].copy()

m5 = m4.copy()
m5[1,1] = 1

vertex_matrices[4] = [m2, m3, m4, m5]
vertex_matrices[8] = [m1, m2.T, m3.T]



cache = []
def is_burl(rep, length):
    result = determine_burl(rep)
    if not result:
        cache[length].add((str(rep[0].tolist()), rep[1]))

    return result

def generate_representations(length):
    if length == 1:
        #If length one, iterate over all of the matrices representing a fiber
        for matrix in vertex_matrices[4]:
            if matrix.shape[1] == 4:
                yield (matrix, 2)
            else:
                yield (matrix, 1)
        for matrix in vertex_matrices[8]:
            if matrix.shape[1] == 4:
                yield (matrix, 1)
            else:
                yield (matrix, 0)
    else:
        #Else find all matrices representing a path of length n-1
        for rep in cache[length-1]:
            matrix, numtwos = rep
            #Convert from string to matrix
            matrix = np.matrix(json.loads(matrix), dtype=np.int32)

            #Permutation matrix
            for followup_perm in perm_matrices[matrix.shape[1]]:
                #Fiber matrix
                for followup_vert in vertex_matrices[matrix.shape[1]]:
                    #Check if this adds a 2-edge-cut in the path
                    if followup_vert.shape[1] == 4:
                        if numtwos < 2:
                            yield (reduce_matrix(matrix*followup_perm*
                                followup_vert), numtwos+1)
                    else:
                        yield (reduce_matrix(matrix*followup_perm*
                            followup_vert), numtwos)
```

```
def generate_representations_pre(length):
    """Exact same functionality as generate_representations(length), but
       multiplies on the other side"""
    if length == 1:
        for matrix in vertex_matrices[4]:
            if matrix.shape[1] == 4:
                yield (matrix, 2)
            else:
                yield (matrix, 1)
        for matrix in vertex_matrices[8]:
            if matrix.shape[1] == 4:
                yield (matrix, 1)
            else:
                yield (matrix, 0)
    else:
        for rep in cache[length-1]:
            matrix, numtwos = rep
            matrix = np.matrix(json.loads(matrix), dtype=np.int32)

            for followup_perm in perm_matrices[matrix.shape[0]]:
                for followup_vert in vertex_matrices[matrix.shape[1]]:
                    if followup_vert.shape[1] == 4:
                        if numtwos < 2:
                            yield (reduce_matrix(followup_vert.T*
                                followup_perm*matrix), numtwos+1)
                    else:
                        yield (reduce_matrix(followup_vert.T*followup_perm*
                            matrix), numtwos)




def reduce_matrix(mat):
    #Take the minimum with a matrix containing twos.
    return np.minimum(mat, np.asmatrix(2*np.ones(mat.shape), dtype=np.int32))

def determine_burl(rep):
    matrix, numtwos = rep

    #First use the simpler checks
    if matrix.shape[1] == 4:
        indices = [[0], [1,3], [2,3]]
    else:
        indices = [[1, 3, 5, 7], [2, 3, 6, 7], [4, 5, 6, 7],
                   [0, 1], [0, 2], [0, 4]]

    for combination in indices:
        allGood = True
        for idx in combination:
            if np.any(matrix[:, idx] == 1):
                allGood = False
                break
        if allGood:
            return True

    if matrix.shape[0] == 4:
        indices = [[0], [1,3], [2,3]]
    else:
        indices = [[1, 3, 5, 7], [2, 3, 6, 7], [4, 5, 6, 7],
                    [0, 1], [0, 2], [0, 4]]


    for combination in indices:
        allGood = True
        for idx in combination:
            if np.any(matrix[idx, :] == 1):
                allGood = False
                break
        if allGood:
            return True

    #If simpler checks are not clear, use the linear program
    if np.any(matrix == 2):
        lprogram = matrix_to_lprogram(matrix)

        f = open ("tmp.lp", 'w')
        f.write(lprogram)
        f.close()
        bound = check_output(["./check_combination.sh", "tmp.lp"])
        if float(bound) >= 0:
            return True


    return False

def matrix_to_lprogram(matrix):
    #Construct the linear program as described in Section 5.4
    lprogram = ""
    for i in xrange(matrix.shape[0]):
        for j in xrange(matrix.shape[1]):
            if i == 0 and j == 0:
                lprogram += "p00"
            else:
                lprogram += "+ p" + str(i) + str(j)
    lprogram += " = 1;\n"

    for i in xrange(matrix.shape[0]):
        lprogram += "p" + str(i) + "0"
        for j in xrange(1,matrix.shape[1]):
            lprogram += "+ p" + str(i) + str(j)
        lprogram += "= row" + str(i) + ";\n"

    for j in xrange(matrix.shape[1]):
        lprogram += "p0" + str(j)
        for i in xrange(1,matrix.shape[0]):
            lprogram += " + p" + str(i) + str(j)
        lprogram += " = col" + str(j) + ";\n"

    if matrix.shape[0] == 4:
        combs = [[1, 3], [2, 3]]
    else:
        combs = [[1, 3, 5, 7], [2, 3, 6, 7], [4, 5, 6, 7]]
    for comb in combs:
        lprogram += "3*row" + str(comb[0])
        for c in comb[1:]:
            lprogram += " + 3*row" + str(c)
        lprogram += " = 1;\n"

    if matrix.shape[1] == 4:
        combs = [[1, 3], [2, 3]]
    else:
        combs = [[1, 3, 5, 7], [2, 3, 6, 7], [4, 5, 6, 7]]
    for comb in combs:
        lprogram += "3*col" + str(comb[0])
        for c in comb[1:]:
            lprogram += " + 3*col" + str(c)
        lprogram += " = 1;\n"

    target = "min:"
    if matrix[0,0] == 1:
        target += "-p00"
    else:
        target += "2*p00"
    for i in xrange(matrix.shape[0]):
        for j in xrange(matrix.shape[1]):
            if i == 0 and j == 0:
                continue
            if matrix[i,j] == 1:
                target += " -p" + str(i) + str(j)
            else:
                target += " +2*p"+str(i)+ str(j)
    lprogram = target + ";\n" + lprogram
    return lprogram


def check_paths(length):
    print length
    non_burl_found = False
    #Generate representations by multiplying both left and right and only
       consider the intersection.
    rep1 = set([(str(rep[0].tolist()),rep[1]) for rep in
       generate_representations(length)])
    rep2 = set([(str(rep[0].tolist()),rep[1]) for rep in
       generate_representations_pre(length)])


    for rep in [(np.matrix(json.loads(rep[0])), rep[1]) for rep in rep1.
       intersection(rep2)]:
        if not is_burl(rep, length):
            non_burl_found = True
    if not non_burl_found:
        print "All paths of length: " + str(length) + " are burls"
        sys.exit(0)

#Set up the actual calculations
count = 0
cache.append(set())
for length in xrange(1, 34):

    cache.append(set())
    check_paths(length)
    print len(cache[length])
```