BSc Thesis Applied Mathematics

Two-Face-Colourable Maps

Chendo Helmink

Supervisor: B. Manthey & J. van Rhijn

June 28, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

I want to thank Bodo Manthey and Jesse van Rhijn for their help in producing this paper.

Two-Face-Colourable Maps

Chendo. E. Helmink*

June 28, 2024

Abstract

We consider the following problem. We are given a plane graph $G = (V, E)$. What is the smallest number of edges that we have to add to G to make it two-face-colourable? We show that a plane graph is two-face-colourable if and only if its inner vertices all have even degree. We present an algorithm that solves this problem in polynomial time.

Keywords: two-face-colourable, maps, face colouring, graph editing, edge addition, plane graphs

1 Introduction

The four-colour theorem is a fundamental theorem within graph theory. The theorem states that every plane map could be coloured so that two faces that border each other are coloured differently by using only four colours. The history of the four-colour theorem goes back to the year 1852, when Francis Guthrie realised he could colour the map of England using only four colours. He soon realised that this worked for every map he could find. It took until 1976 before Kenneth Appel and Wolfgang Haken produced a valid proof. It was the first major theorem to be proved by a computer [1].

Since then, graph coloring grew into a large field within graph theory [6]. An example is sports scheduling, where edge coloring plays a big role [7]. In a lot of cases it can be suprising that graph coloring can provide a solution. An example is solving Sudokus [10].

These graph coloring examples all look at the properties of a graph. There are also fields within graph theory that look into modifications to graphs to obtain certain properties. This field of graph theory is called graph editing. A famous family of graphs are Eulerian graphs. All vertices in an Eulerian graph have even degree. An example of graph editing is the article of F.T. Boesch, C. Stuffel and R. Tindell called "The Spanning Subgraphs of Eulerian Graphs" [2]. This article describes an algorithm that finds the smallest number of edges to add to a graph to make the graph Eulerian. Another paper about graph editing related to Eulerian graphs is the paper by Konrad K. Dabrovski, Petr A. Golovach, Pim van 't Hof and Daniël Paulusma called "Editing to Eulerian graphs" [5].

There are no efficient algorithms known yet to check if a graph is three-face-colourable and it is possible that such an algorithm does not even exist. However, checking for a graph to be two-face-colourable is efficiently possible. If a graph is Eulerian, then the graph is two-face-colourable. By allowing certain operations, every plane graph can be made two-face-colourable. This paper focuses on achieving two-face-colourability by

*Email: c.e.helmink@student.utwente.nl

adding edges. We present an algorithm that finds a smallest set of edges to add to a graph to make it two-face-colourable.

One of the fields where two-face-colourable maps can be of importance is livestock farming. For example, bulls and boars (male pigs) are both known to become aggressive when they see a male of the same species. If one were to build the pens of these animals in such a way that a bear does not see another bear and a bull does not see another bull, this could reduce aggression.

2 Preliminary

This part of the paper explains some of the concepts that are used within the rest of the paper.

2.1 Basic terminology and notation

In this paper we look at *planar* graphs. A graph is said to be planar if it can be drawn in the plane so that its edges only intersect in their ends. A *plane graph* is a drawing of a planar graph such that its edges only intersect in their endpoints. Such a drawing is called an *embedding*. If $G = (V, E)$ is a plane graph, then G divides the plane into connected regions which are called *faces*. The face with unbounded area is called the *outer face*. A *face-dividing edge* we define to be an edge between two vertices within a face, such that the face gets divided into two new faces. Two faces are *adjacent* if they share an edge.

We focus on *two-face-colourable* graphs. A graph is two-face-colourable if its faces can be coloured with two distinct colours in such a way that if two faces are adjacent, they receive distinct colours. A plane graph that is two-vertex-connected, we call a *map*. When the embedding of a graph is fixed, we can divide the vertices of the graph into *outer vertices* and *inner vertices*. Outer vertices are adjacent to the outer face and inner vertices are not adjacent to the outer face. The *parity* of a vertex is even when a vertex has even degree and odd when a vertex has odd degree.

Given a plane graph G, one can define another graph G^* as follows: Corresponding to each face f of G there is a vertex v^* of G^* and corresponding to each edge e of G there is an edge e^* of G^*; two vertices v_1^* and v_2^* are joined by the edge e^* in G^* if and only if their corresponding faces f and g are separated by the edge e in G. The graph G^* is called the *dual* of G. The dual of a graph is planar [3, Chapter 9.2].

A graph H is a *subgraph* of G (written $H \subseteq G$) if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Similarly, a graph G_S is called a supergraph of G if and only if G is a subgraph of G_S. A subset M of E is called a *matching* in G if no edges in M are adjacent in G. A matching that covers every vertex of G is called a *perfect matching*.

From now on, $G = (V, E)$ denotes a map G, with the set of vertices V and the set of edges E, where $|V| = n$ and $|E| = m$. The set of faces of a map we denote by F. The dual of a map G we denote by G^*.

3 Two-face-colourable maps

A plane graph is called two-face-colourable if every face of the graph can be coloured in such a way that no two faces that are adjacent to each other have the same colour and the number of distinct colours is at most two. Not every graph has this property.

An example of a plane graph that is not two-face-colourable is shown in Figure 1.

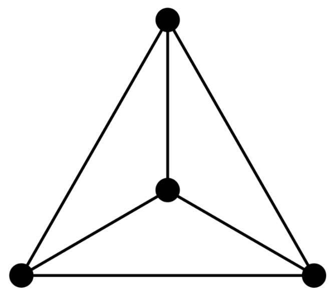

FIGURE 1: Example of a plane graph that is not two-face-colourable

One possible way to obtain a plane map that is two-face-colourable is by adding edges to a map G. We focus on the following research question.

Research Question: *Given a map $G = (V, E)$, what is the smallest $k \in \mathbb{Z}$ such that there exists a set D of k face-dividing edges with $G' = (V, E \cup D)$ is two-face-colourable?*

3.1 Assumptions, observations and restrictions

For this bachelor's thesis, we look at plane graphs and therefrom follow some restrictions.

The first restriction is that we can only add edges that do not intersect each other except from their endpoints. This restriction follows from the fact that if edges intersect, then the embedding of the graph is not planar anymore. We are only allowed to add edges within inner faces. We chose to do so as otherwise a vertex that was an outer vertex in the original graph, could become an inner vertex after adding an edge. We do allow multiple edges between vertices since otherwise not every graph can be made two-face-colourable by adding edges. All maps G can be made two-face-colourable by duplicating every edge in G since this results in an Eulerian map.

We assumed that the embedding of the graph is given. This makes the problem well-defined as it is now clear which edges we are allowed to add to the graph and which face is the outer face. The edges which we are allowed to add to a graph, we call the *potential edges.* Since the set of potential edges of a graph is easy to determine, we assume this set of potential edges to be given as input. Potential edges that lie within distinct faces are considered distinct edges.

We also assume that the graph is two-vertex-connected. If the graph is not two-vertex-connected, then we can divide the faces into two subsets, such that for every face in the first set, none of the faces of the second set are adjacent to that face. We can look at these two subsets separately.

Since we only consider maps, having loops in our graph makes no sense. Therefore we consider all graphs to be loopless graphs.

3.2 Properties

If within a map there exists an inner vertex that is adjacent to an odd number of faces then this graph is not two-face-colourable. This shows that if a map is two-face-colourable, then all inner vertices have even degree. We need this to prove Theorem 3. We include a simple proof.

Theorem 1. *If a map is two-face-colourable, then all inner vertices have even degree.*

Proof. Let G be a map. Let there exist at least one inner vertex with odd degree. Let v_0 be an inner vertex with odd degree, where its degree is denoted by d_0. Since all maps are two-vertex-connected, $d_0 \geq 3$. Let $f_1, ..., f_{d_0}$ be the faces adjacent to v_0. Let G_0 be the subgraph of G that consists only of the vertices and edges of the faces $f_1, ..., f_{d_0}$. The dual (without the outer face of G_0) corresponding to G_0 is a cycle of length d_0. Since $d_0 \geq 3$ and d_0 is odd, the chromatic number of the dual is at least 3. This implies that the chromatic number of G_0 is at least 3 as well [3, Page 158]. The graph G_0 is therefore not two-face-colourable. Since G_0 is not two-face-colourable, the supergraph G is not two-face-colourable either. This shows that if a graph is two-face-colourable, then all inner vertices have even degree. $\square$

From now on we consider the number of faces that a vertex is adjacent to, to be the number of faces minus the outer face. This comes from the fact that when colouring a graph, we decide to only assign colours to the faces that are not the outer face. If we would assign a colour to the outer face, then the only graphs that are two-face-colourable are graphs that consist of only faces that all completely lie within another face (except the outer face). In our case we are colouring a map. The outer face is also not part of the map itself and therefore it does not make sense to colour the outer face.

Let $G = (V, E)$ be a plane graph. Let f_1 and f_2 be two faces in G that share an edge e. Let $G^* = (V^*, E^*)$ be a dual of G. Let v_1^* and v_2^* be the vertices in G^* that represent f_1 and f_2 respectively. Since f_1 and f_2 share an edge, there exists an edge e^* in G^* with v_1^* and v_2^* as endpoints. Since faces are connected and both f_1 and f_2 contain e, we can draw a line from v_1 to a point p on the edge e and a line from v_2 to p, such that the edge consisting of these two lines combined, intersects with e and has v_1 and v_2 as endpoint. Since the faces are connected and have no edges of G within the area enclosed, e does not intersect any other edges in G. Therefore we can always draw a dual of a graph in such a way that every edge uv in G^* only intersects with the shared edge of the corresponding faces f and g in G. From now on we assume that every dual of a graph is drawn in such a way.

This way of drawing the dual helps us to prove the next theorem. We need this theorem to prove Theorem 3.

Theorem 2 (From Graph theory with applications [3, Exercise 9.2.3])**.** *If a plane map G is Eulerian then G^* is two-colourable.*

Proof. Let G be a plane map. Assume G^* is not two-colourable. This implies that G^* is not bipartite. If a graph is not bipartite then it contains an odd cycle so G^* contains an odd cycle C.

Let F_C be the set of faces of G that correspond to the vertices in C. Since C is odd, C contains of an odd number of edges. An edge between two vertices in G^* implies that the two faces in G corresponding to these two vertices share an edge. Since C contains an odd number of vertices, F_C contains of an odd number of faces.

Since every edge in C intersects exactly one shared edge of faces F_C and C contains an odd number of edges, there are an odd number of edges contained in the faces of F_C that intersect C.

Let E_{cross} be the set of edges in G that intersect C. Note that every edge in C corresponds to exactly one edge in E_{cross}. Since C contains an odd number of edges, E_{cross} contains an odd number of edges as well. Since C is a closed cycle, C encloses an area in G.

Let V_{in} be the vertices that lie within the area enclosed by C. Let E_{in} be the set of edges between vertices in V_{in}.

For $v \in V_{\text{in}}$ let $\deg(v) = d_{\text{in}}(v) + d_{\text{cross}}(v)$. Here $d_{\text{in}}(v)$ is the number of edges in E_{in} that has v as an endpoint and $d_{\text{cross}}(v)$ is the number of edges in E_{cross} that has v as an endpoint. Let s_d be the sum of the degrees of the vertices V_{in}. Since all edges in E_{in} have two endpoints in V_{in}, $\sum_{v \in V_{\text{in}}} d_{\text{in}}(v) = 2k$, with $k \in \mathbb{N}$. The edges in E_{cross} all have exactly one endpoint within V_{in}. Therefore $\sum_{v \in V_{\text{in}}} d_{\text{cross}}(v) = l$, with l an odd positive integer. Since $s_d = \sum_{v \in V_{\text{in}}} d_{\text{in}}(v) + d_{\text{cross}}(v) = 2k + l$, we have that s_d is an odd number. Since s_d is odd, there must exist at least one vertex v_0 in V_{in} such that $\deg(v_0)$ is odd.

Since G contains an odd vertex, G is not Eulerian. Thus, we have shown that if G^* is not two-colourable then G is not Eulerian. This implies that if G is Eulerian, then G^* is two-colourable. □

We have now given a proof that works for Eulerian plane graphs. All vertices in Eulerian graphs have even degree. The maps we are looking at do not have this property. The proof of the next theorem shows why even parity for all inner vertices is sufficient and necessary for a map to be two-face-colourable.

Theorem 3. *A map is two-face-colourable if and only if all inner vertices have even degree.*

Proof. If a map is two-face-colourable, then all inner vertices have even degree by Theorem 1.

Let G be a map such that all inner vertices have even degree. The sum of the degrees of all inner vertices is even. Since the degree sum of all vertices must be even, the degree sum of the outer vertices must be even as well. This implies that there are an even number of odd outer vertices.

Let C be the cycle of outer vertices. Let v_0 be an outer vertex with odd degree. Walk around C clockwise until we find another vertex v_1 with odd degree. Connect v_0 to v_1 with an edge via the outer face. Continue walking around C until we find two more vertices with odd degree and connect them with an edge via the outer face as well.

Since there are an even number of odd outer vertices, we can continue this process until all odd outer vertices are connected to another odd outer vertex via an edge. None of these edges intersect, so the resulting graph G_1 is still a plane graph.

Since all vertices of G_1 are even, G_1 is Eulerian. By Theorem 2 the dual G_1^* is two-colourable. Note that adding edges in the outer face adds new vertices in G_1^* compared to G^*. All edges and vertices in G^* still exist in G_1^*. Therefore the chromatic number of G^* is smaller or equal to the chromatic number of G_1^*.

Since G_1^* is two-colourable, G^* is two-colourable. Since G^* is two-colourable, G is two-face-colourable. □

We have now showed that to answer our research question, we can look at the following. *Given a map $G = (V, E)$, what is the smallest $k \in \mathbb{Z}$ such that there exists a set D of k face-dividing edges with all inner vertices in $G' = (V, E \cup D)$ having even degree?*

4 Algorithm

We design an algorithm to compute the smallest number of edges k that we have to add to a graph to make the graph two-face-colourable. This algorithm should be able to compute the value k in a time that is polynomial in $|V|$. Since the embedding of the graph is given, we also know which vertices are the outer vertices of the graph.

By Theorem 3, to make a graph two-face-colourable it is necessary and sufficient that all inner vertices have even degree. Let v_0 and v_k be two distinct odd inner vertices. If we add edges v_0v_1,v_1v_2,...,$v_{k-1}v_k$ to create a path $v_0v_1....v_{k-1}v_k$ between two odd inner vertices v_0 and v_k, then the degree of both odd inner vertices increases by one and they become even inner vertices. All other vertices that lie on the path do not change parity since they are not endpoints of the path and therefore two edges of the path are adjacent to such a vertex.

We can also add a path from an odd inner vertex to an outer vertex. As shown before, the degrees of the outer vertices do not play a role for our problem. By adding a path between an odd inner vertex and an outer vertex, the degree of the odd inner vertex increases by one. The degree of the outer vertex also increases by one. The degrees of all other vertices on the path increase by two.

The *potential graph* G_P is the graph consisting of vertices V and the set of potential edges E_P. With *potential distance* between two vertices we mean the distance between these vertices in the *potential graph*. A *potential path* is a shortest path of potential edges between two vertices.

4.1 Finding distance between odd inner vertices

We now have to determine for every odd inner vertex how many edges we have to add to create a new path between this vertex and every other odd inner vertex. We also have to find how many edges we have to add to create a new path from this odd inner vertex to the boundary of the graph. We now want to compute the potential distance between all odd inner vertices.

Since the set of potential edges of our graph is given, we can compute the shortest potential path between two vertices. The potential distance between two odd inner vertices is the length of the shortest potential path between two odd inner vertices. To find the potential distances between vertices we make use of the breadth-first search algorithm [8, Chapter 5.1]. This algorithm is often used to find the shortest distance between two vertices.

4.2 Adding edges

Now that we have found the potential distances between all odd inner vertices and the potential distances to the boundary of these odd inner vertices, we have to determine which edges to add to our original graph. For all odd inner vertices we either connect them to another odd inner vertex or to a boundary vertex.

Note here that when connecting an odd inner vertex v_0 to another odd inner vertex v_1, this not only changes the parity of v_0 from odd to even, but also the parity from v_1 from odd to even. When an odd inner vertex is connected to the boundary, only one odd inner vertex changes parity from odd to even.

4.3 Finding smallest number of edges to add to our graph

To find the smallest number of edges k that we have to add to our graph, we have to find a minimum weight perfect matching [9, Page 262]. We set up a *matching graph.* The graph $G_M = (V_M, E_M)$ we denote as the matching graph. The set of vertices V_M of our matching graph consists of all odd inner vertices and a copy of all these vertices that all represent the boundary. The set of edges E_M of our matching graph consists of edges between every pair of vertices in V_M.

The cost of matching two odd inner vertices is equal to the potential distance between these vertices. The cost of matching an odd inner vertex with a boundary representing vertex is equal to the potential distance to the closest outer vertex. The cost of matching two boundary representing vertices is zero, since the degree of outer vertices is not of importance to us and therefore we never add a potential path between boundary vertices. When we match two vertices, we add the edges of a shortest potential path between the two vertices to our set of added edges. It could occur that added edges intersect each other. This causes our graph to not be planar anymore. We rearrange the edges in a way that they do not intersect.

We first run BFS. The total running time of BFS is $O(|V| + |E|)$ [4, Page 534]. The weighted minimum perfect matching algorithm has total running time of $O(|V|^4)$ [9, Page 261]. Every vertex in a face is adjacent to at most one potential edge. Therefore rearranging the edges has a total running time $O(|V| \cdot |F|)$. We conclude that our algorithm computes a solution in polynomial time in $|V|$.

4.4 Pseudocode

In the following pseudocode, we use several symbols. The symbols which have not been introduced earlier are defined as follows.

With p we denote $|V_M|$. We define *added edges* to be potential edges that are added to our graph. The $\text{cost}(v_i, v_j)$ equals the potential distance between v_i and v_j and is the cost of using an edge $v_i v_j$ in the graph G_M. Since the potential distance between v_i and v_j is the same as between v_j and v_i, $\text{cost}(v_i, v_j) = \text{cost}(v_j, v_i)$. The potential distance between two vertices v_0 and v_1 we abbreviate with $\text{pd}(v_0,v_1)$. A set of vertices that are adjacent to an added edge within a face f we denote by S_{af}. We denote the set of edges contained in the shortest potential paths between vertices that are matched by our algorithm by D_M.

The input of our algorithm is a plane map $G = (V, E)$ with faces F.

Algorithm 1 Finding optimal solution

for all odd inner vertices v in V **do**
    Let BFS run in G_P from v to find pd(v,u) for all vertices $u \neq v$ and compute a sp(v,u).
end for

Create graph G_M with E_M and V_M empty
Create empty set D_M
Let $V_M = v_1, v_2, \ldots, v_p \cup b_1, b_2, \ldots b_p$
for $i = 1, 2, \ldots, p$ **do**
    for $j = i+1, i+2, \ldots, p$ **do**
        Add edge $v_i v_j$ to E_M with $\text{cost}(v_i, v_j) = \text{pd}(v_i, v_j)$ in G
        Add edge $b_i b_j$ to E_M with $\text{cost}(b_i, b_j) = 0$
    end for
    Add edge $v_i b_i$ to E_M
    Find outer vertex b^* with smallest potential distance from v_i.
    $\text{cost}(v_i, b_i) = \text{pd}(v_i, b^*)$.
end for

Find a minimum weight perfect matching M for G_M

for all edges $e \in E_M$ **do**
    Add all edges from sp(e) to D_M
end for

Run Algorithm 2

Algorithm 2 Rearranging edges

for all faces f in G **do**
    $i = 0$
    for all v in f clockwise **do**
        if v is adjacent to an added edge within f **then**
            $i = i + 1$
            Add v to S_{af} with label i.
        end if
    end for
    Remove all added edges within f from G
    while S_{af} is not empty **do**
        for two vertices (u,v) with lowest label in S_{af} **do**
            Add edge uv to G
            Remove u,v from S_{af}
        end for
    end while
end for

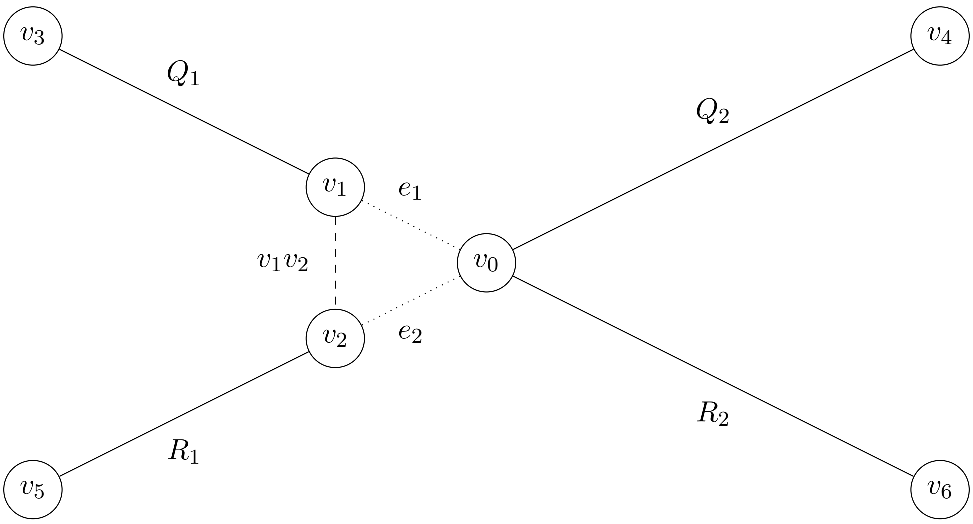

FIGURE 2: The path P_1 is $v_3Q_1v_1v_0Q_2v_4$. The path P_2 is $v_5R_1v_2v_0R_2v_6$. The path P_3 is $v_3Q_1v_1v_2R_1^*v_5$ and the path P_4 is $v_4Q_2^*v_0R_2v_6$. The vertices v_0 until v_6 could be nondisjoint and Q_1, Q_2, R_1 and R_2 could be empty.

5 Proof of correctness

This section provides a proof that our algorithm returns an optimal solution. For this section we need some definitions. We define an *added path* to be a path consisting of added edges. We define E_a to be the set of potential edges that is added to our graph by the algorithm before rearranging the edges.

We first give a proof that the algorithm returns a feasible solution. By Theorem 3 our final graph needs to be a plane graph and all inner vertices need to obtain even degree. Algorithm 1 computes a set of edges E_a that can intersect. We show that these edges can be rearranged such that they do not intersect, while returning a set of edges with same cost as E_a and not changing degrees of vertices in G.

The idea of rearranging the edges is as follows. Within a face we look at all vertices that are adjacent to an added edge within that face. We remove the edges between all vertices within the face. The vertices that were adjacent to an added edge within the face we connect to each other pairwise and clockwise such that all these vertices are still adjacent to exactly one added edge within the face. To prove that this works we need that all vertices are adjacent to at most one added edge within the face and that there are an even number of vertices within the face that are adjacent to an added edge. The following two lemmas are needed to prove Lemma 6.

We introduce a new notation for following a path in opposite direction. Let P_1 be the path from v_0 to v_1. The opposite path from v_1 to v_0 we denote by P_1^*. A drawing is included in Figure 2 to get a better image of the next proof.

Lemma 4. *Within E_a, all vertices are adjacent to at most one added edge within a face.*

Proof. We give a proof by contradiction. Let M be the minimum perfect matching found by our graph. Let S_M be a set of shortest potential paths between the matched vertices. Recall that E_a is the set of edges that are contained in S_M.

Let f_1 be a face that contains a vertex v_0 that is adjacent to two added edges e_1 and e_2 in f_1. Let v_1 and v_2 be the second endpoints of e_1 and e_2 respectively. There exists paths P_1 and P_2 in S_M such that e_1 lies on P_1 and e_2 lies on P_2. We distinguish two cases.

The first case is that P_1 is the same path as P_2. Since both e_1 and e_2 lie on P_1, part of P_1 is $v_1v_0v_2$. However, replacing this part of P_1 by v_1v_2 lowers the number of edges on the path and does not change the endpoints of the path. This contradicts that P_1 is a shortest potential path.

The second case is that P_1 and P_2 are two distinct paths.

Let v_3 and v_4 be the endpoints of P_1. Let v_5 and v_6 be the endpoints of P_2. Let P_1 be $v_3Q_1v_1v_0Q_2v_4$ such that Q_1 contains all edges between v_3 and v_1 on P_1 and Q_2 contains all edges between v_0 and v_4 on P_1. Let P_2 be $v_5R_1v_2v_0R_2v_6$, with R_1 the edges between v_5 and v_2 and R_2 the edges between v_0 and v_6.

Let P_3 be $v_3Q_1v_1v_2R_1^*v_5$. Let P_4 be $v_4Q_2^*v_0R_2v_6$. The paths P_1 and P_2 correspond to a matching of v_3 to v_4 and v_5 to v_6 in the matching graph. The paths P_3 and P_4 correspond to a matching of v_3 to v_5 and v_4 to v_6 in the matching graph. In both cases all four vertices are matched. See Figure 2 for an image of this situation.

The paths P_1 and P_2 together consists of the same edges as P_3 and P_4 together, except that in the second case e_1 and e_2 are removed and v_1v_2 is added. Since P_3 and P_4 together contain of one edge less than P_1 and P_2 together, the cost of matching v_3 to v_5 and v_4 to v_6 is at least one less than matching v_3 to v_4 and v_5 to v_6.

Note that it could happen that this new matching matches two boundary vertices to each other, matching such two vertices gives a cost of zero in the matching graph so this still gives a lower cost. Matching v_3 to v_5 and v_4 to v_6 gives a matching with lower cost than M. This contradicts that M is a minimum perfect matching.

Note that P_1 matches (v_3,v_4), P_2 matches (v_5,v_6), P_3 matches (v_3,v_5) and P_4 matches (v_4,v_6). Let t_i (v,u) be the edge in M such that P_i matches v and u.

We have not made any assumptions on v_0, v_3, v_4, v_5 and v_6 being distinct. We now look at the cases where some of these vertices are the same. Note that v_0 can not be the same vertex as v_3 and v_5 since v_1 lies on the path between v_3 and v_0 and similarly v_2 lies on the path between v_5 and v_0. Since both paths P_1 and P_2 have distinct endpoints, v_3 can not be the same vertex as v_4 and v_5 can not be the same vertex as v_6.

The first case is that v_3 and v_6 are the same vertex. In this case we can replace v_6 by v_3 in our matching. In this case we get t_2 (v_5,v_3) and t_4 (v_4,v_3). All vertices are still matched the same number of times and no vertices are matched to themselves so nothing changes in our reasoning. The case $v_4 = v_5$ has the same reasoning.

The second case is that v_4 is the same vertex as v_6. In this case we can replace v_6 by v_4 in our matching. In this case we get t_2 (v_5,v_4) and t_4 (v_4,v_4). Note that v_4 was matched twice via both P_1 and P_2. This implies that v_4 is a boundary vertex. We now have the case that v_4 is matched to itself. Since v_4 is a boundary vertex, it is represented by two distinct boundary representing vertices in G_M. We can match these two distinct boundary vertices to each other with cost zero. This still yields a perfect matching with lower cost than M so our reasoning still holds.

We now still remain with the case that v_4 or v_6 equals v_0. If exactly one of the vertices v_4 and v_6 is the same as v_0 we just replace v_4 or v_6 by v_0 and the rest of our reasoning remains the same.

If both v_4 and v_6 are equal to v_0 then we get the following sub case of the second case. We have t_1 (v_3,v_0), t_2 (v_5,v_0) and t_3 (v_3,v_5). In the case that $v_0 = v_4 = v_6$, both Q_2 and R_2 are empty. This is the only case that causes one of the paths P_1 until P_4 to be empty, namely P_4. If a path is empty then that path simply does not exist. Therefore after replacing P_1 and P_2 by P_3 we have that v_0 was matched twice, but is now not matched anymore.

Since v_0 was an endpoint of two paths, we have that v_0 is a boundary vertex. Therefore v_0 is represented by two distinct boundary vertices. We can match these boundary representing vertices to each other with cost zero and the rest of our reasoning is the same as the second case.

We have now looked at all cases. We conclude that within E_a, all vertices are adjacent to at most one added edge within a face. □

From Lemma 4, we can easily get to the following lemma.

Lemma 5. *Within E_a, every face has an even number of vertices that is adjacent to an added edge within that face.*

Proof. By Lemma 4, every vertex within a face is adjacent to at most one added edge within that face. Therefore no added edges share an endpoint within a face. Since every added edge has two endpoints, the total number of vertices that is adjacent to an added edge within a face is even. □

The following lemma shows that we can rearrange the edges such that they do not intersect.

Lemma 6. *It is always possible to rearrange the edges of E_a, such that no edges intersect and all vertices are still an endpoint of the same number of edges.*

Proof. Recall that all our added edges lie within a face. By Lemma 4 a vertex is never adjacent to more than one added edge within a face. Therefore all vertices within a face, are either adjacent to one added edge or to zero added edges within that face.

By Lemma 5, there are an even number of vertices within a face that are adjacent to an added edge. Let V_{face} be the set of vertices within a face f_1 that are adjacent to an added edge within f_1. Let $q = |V_{\text{face}}|$. Start at one vertex within the face that is adjacent to an added edge within f_1 and call this vertex v_1. Now walk around the vertices of the face f_1 clockwise and call the next vertex, that is adjacent to an added edge, v_2 and so forth until we arrive at vertex v_q.

The set of edges X could have two added edges that intersect in f_1. If we remove the added edges of X that lie within f_1 and replace them with the following new added edges, we could make sure that no two added edges intersect anymore: add an added edge between v_1 and v_2, between v_3 and v_4 and so forth until v_{q-1} and v_q. Still all vertices v in V_{face} are adjacent to exactly one added edge. Therefore this returns a set of edges that has the same cost as E_a, contains no intersecting edges and does not change the degree of vertices. $\square$

We now give a proof that our algorithm computes a set of edges such that in the resulting graph all inner vertices have even degree and show that our algorithm computes a feasible solution.

Lemma 7. *All inner vertices have even degree after adding the edges in E_a.*

Proof. Let M be the minimum perfect matching found by our graph. Let S_M be a set of shortest potential paths between the matched vertices. Recall that E_a is the set of edges that are contained in S_M. All odd inner vertices are an endpoint of exactly one path in S_M. All even inner vertices are not an endpoint of a path in S_M. Outside of these added paths, no other edges are added to our graph. Therefore all odd inner vertices change parity and all even inner vertices do not change parity. $\square$

Theorem 8. *Algorithm 1 computes a feasible solution.*

Proof. By Lemma 7 all inner vertices have even degree. By Lemma 6 no edges intersect. Therefore our algorithm computes a plane graph. By Theorem 3 these two properties are sufficient for a graph to be two-face-colourable. We conclude that the algorithm computes a feasible solution. $\square$

We now show that Algorithm 1 computes an optimal solution. We do this by showing that an optimal solution can be partitioned into paths such that we can compare these paths with the paths between the matched vertices in our matching graph. To perform the partitioning into paths, we first have to show that an optimal solution is acyclic.

Lemma 9. *An optimal solution contains no cycles.*

Proof. Let X be an optimal solution. Let E_X be the set of edges that X contains. Assume that there exist edges e_1 until e_c in E_X such that e_1 until e_c form a cycle C. All vertices in a cycle have degree 2. Therefore deleting e_1 until e_c from E_X does not change the parity of any vertex. Therefore removing e_1 until e_c from E_X gives a solution with lower cost than X. This contradicts that X is an optimal solution. $\square$

We now show how an acyclic solution can be partitioned into paths with desired properties. These properties are the endpoints of the paths and the edge-disjointness of the paths.

Theorem 10. *Given an acyclic solution X there exists a collection of paths P_P such that this collection contains the same edges as X, with all odd inner vertices in G being an endpoint of a path in P_P exactly once, all even inner vertices in G not being an endpoint of a path in P_P and all paths in P_P being edge-disjoint.*

Proof. Let X be an acyclic solution. Let E_X be the set of edges that X contains. Let P_X be the set of paths such that every path consists of one edge of E_X and every edge is contained in exactly one path.

Let v_0 be an inner vertex that is an endpoint of at least two paths in P_X. Let v_1 and v_2 be endpoints of two paths with endpoint v_0. Since X is acyclic, the vertices v_1 and v_2

are distinct. We remove the paths $v_0 \dots v_1$ and $v_0 \dots v_2$ from P_X and replace them with the path $v_1 \dots v_0 \dots v_2$ containing the same edges as contained in the paths $v_0 \dots v_1$ and $v_0 \dots v_2$ together. This reduces the number of paths with v_0 as endpoint by two.

Since every edge in E_X is contained in exactly one path in P_X, P_X was edge-disjoint at the start of this process. During our process, every time we remove paths and replace them with a new path, the new path consists of the exact same edges as the paths that get removed. During this process replacing paths in P_X is the only operation. Therefore P_X is still edge-disjoint after replacing paths by new paths.

Since even inner vertices are an endpoint of an even number of paths and odd inner vertices are an endpoint of an odd number of paths in P_X, we can continue this process for all inner vertices until all even inner vertices are an endpoint of zero paths in P_X and all odd inner vertices are an endpoint of exactly one path in P_X. □

We now show that Algorithm 1 computes an optimal solution.

Theorem 11. *Algorithm 1 computes an optimal solution.*

Proof. Let X be an optimal solution. By Lemma 9 an optimal solution is acyclic. By Theorem 10, the solution X can be partitioned into paths, such that the paths are edge-disjoint, all odd inner vertices are an endpoint of exactly one path and all even inner vertices are not an endpoint of a path. After partitioning X into paths, if two odd inner vertices are connected by a path in the potential graph, then we match them in the matching graph. If an odd inner vertex is connected by a path to the boundary then we match the odd inner vertex to a boundary representing vertex in the matching graph.

The cost of matching two vertices in the matching graph equals the shortest potential path between these vertices. Since the potential paths created by partitioning X into paths have length at least as large as the shortest potential path, the total number of edges in X equals at least the cost of the matching M_1 that matches all endpoints of the paths created by partitioning X into paths.

Let $\text{cost}(M_1)$ be the cost of matching M_1. There are an even number of boundary representing vertices that are unmatched. Matching these boundary representing vertices pairwise to each other does not increase the cost. Matching these boundary vertices, together with M_1 gives a perfect matching. Let $\text{cost}(M_2)$ be the cost of matching M_1, together with matching all unmatched boundary vertices. Since matching boundary vertices does not increase the cost, $\text{cost}(M_2) = \text{cost}(M_1)$.

Let M^* be a minimal perfect matching. Since every perfect matching has cost of at least $\text{cost}(M^*)$, we have that $\text{cost}(M_2) \geq \text{cost}(M^*)$. Let $\text{cost}(\text{Alg})$ be the cost of the solution found by our algorithm. Since the algorithm computes a minimal perfect matching, $\text{cost}(\text{Alg}) = \text{cost}(M^*)$. We conclude that $\text{cost}(\text{Alg}) \leq \text{cost}(X)$. By Theorem 8 our algorithm computes a feasible solution. We conclude that our algorithm computes an optimal solution. □

6 Conclusion and discussion

We have described an algorithm that finds a smallest set of edges such that adding these edges results in a two-face-colourable map. The algorithm finds this solution in polynomial time in $|V|$. For further research one can consider an algorithm that finds the smallest number of edges that have to be removed from a map to make it two-face-colourable. Since one can always reduce the number of faces in a map to at most two by removing edges, there always exists such a number. Designing an algorithm that finds the smallest number of operations to make a map two-face-colourable can also be interesting. We considered edge addition in this paper and we already discussed edge removal. One can consider the problem if we allow both edge removal and edge addition.

A relevant paper is the earlier mentioned paper "Editing to Eulerian graphs" [5, Chapter 3] by Dabrovski et al. Part of this article is about edge addition and edge removal. Their question is if given a positive integer k, it is possible to perform k operations to end up with an Eulerian graph. They have described an algorithm that finds an answer to this question in polynomial time, when only edge addition and edge removal are allowed. The maps we considered are not Eulerian so it is possible that no such algorithm exists for two-face-colourable maps, when allowing edge addition and removal. For further research we recommend to take a look into this problem.

References

[1] Kenneth Appel and Wolfgang Haken. The Solution of the Four-Color-Map Problem. *Scientific American*, pages 108–121, 1977. Publisher: Scientific American, a division of Nature America, Inc.

[2] F. T. Boesch, C. Suffel, and R. Tindell. The spanning subgraphs of eulerian graphs. *Journal of Graph Theory*, pages 79–84, 1977.

[3] J. A. Bondy and U. S. R. Murty. *Graph theory with applications.* North Holland, 1976.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction To Algorithms.* MIT Press, 2001.

[5] Konrad K. Dabrowski, Petr A. Golovach, Pim van 't Hof, and Daniël Paulusma. Editing to Eulerian graphs. *Journal of Computer and System Sciences*, pages 213–228, March 2016.

[6] Anjali Gangrade, Bhawna Agrawal, Sanjeet Kumar, and Akhlak Mansuri. A study of applications of graph colouring in various fields. *International Journal of Statistics and Applied Mathematics*, pages 51–53, March 2022.

[7] Tiago Januario, Sebastián Urrutia, Celso C. Ribeiro, and Dominique de Werra. Edge coloring: A natural model for sports scheduling. *European Journal of Operational Research*, pages 1–8, October 2016.

[8] Andreas Paffenholz. Algorithmic Discrete Mathematics. 2021.

[9] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Courier Corporation, January 1998.

[10] Deepika Rai, N.S. Chaudhari, and Maya Ingle. An Efficient Algorithmic 3-SAT Formulation for Sudoku Puzzle using Graph Coloring. In *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*, pages 1–6, December 2018.