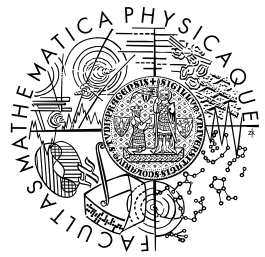


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Eldar Urmanov

Coloring of triangle-free graphs on torus

Mathematical-Physical Faculty

Supervisor of the bachelor thesis: prof. Mgr. Zdeněk Dvořák, Ph.D.
Study programme: Informatics
Study branch: Informatics

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

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

In ..... date ....

Author's signature

Dedication. I am grateful to my parents for providing me with the opportunity to become a student in Charles University and for stoically accepting my decision to study Theoretical Informatics. Also I would like to thank my supervisor for trusting me such a great and important topic and patiently pointing out all occurrences of the same mistakes I have made in the drafts just to see these mistakes appear in the new chapters. A word of gratitude towards my friends and colleagues motivating me to finish my studies in time, advocating for Charles University and translating the abstract of this thesis to Czech language.

Title: Coloring of triangle-free graphs on torus

Author: Eldar Urmanov

Department of Applied Mathematics: Mathematical-Physical Faculty

Supervisor: prof. Mgr. Zdeněk Dvořák, Ph.D., Department of Applied Mathematics

Abstract: Pekárek and Dvořák (2021) proposed a linear-time algorithm to decide 3-colorability of triangle-free graphs drawn on the torus. We implemented this algorithm efficiently and evaluated its performance on a natural class of graphs.

Keywords: toroidal graph 3-colorability algorithm implementation

Contents

Introduction

By a well-known theorem of Grötzsch [1], every triangle-free planar graph is 3-colorable. This result has been proved in a number of different ways [2, 3] and there is a linear-time algorithm to find a 3-coloring of a given triangle-free planar graph [4]. There are infinitely many triangle-free minimal non-3-colorable graphs that can be drawn in any other surface, e.g., Mycielsky graphs of odd cycles. Dvořák, Kráľ and Thomas [5] gave a linear-time algorithm to decide 3-colorability of a triangle-free graph drawn in any fixed surface, however, their algorithm is not practical. Dvořák and Pekárek [6] gave an exact characterization of 3-colorability of triangle-free graphs on torus, and in [7] they designed a practical algorithm to decide whether a given triangle-free toroidal graph is 3-colorable.

The goal of this work is the implementation and evaluation of the algorithm [7] deciding whether a triangle-free toroidal graph is 3-colorable ("the implementation" or "the program" from now on). A graph G is *3-colorable* if there exists a mapping $c : V(G) \to \{0, 1, 2\}$ such that $c(u) \neq c(v)$ holds for every $uv \in E(G)$.

The program receives a graph which does not contain any cycles of length ≤ 3 as subgraphs, is connected and is embeddable on torus without crossings with a correct embedding provided with the graph. The output of the program is binary, indicating whether there exists a correct 3-coloring of the graph or not. Note that the program does not find the coloring but only decides on its existence.

The time complexity of the algorithm used for this problem is $O(|V(G)|)$ but some parts of the implementation use structures like hashmaps and self-balancing BSTs so the actual complexity will be higher. The program could have been written in linear time but the choice was made in favor of code simplification and occasional real execution time reduction.

The program was widely tested on the graphs from [6] and their various augmentations and for the case of small graphs compared with a bruteforce solution. The graphs [6] from with non-floating faces arbitrarily quadrangulated are non-3-colorable, so the comparison with bruteforce was not needed. The program passed all tests correctly and the benchmarks showed that the execution time is practically linear to the input size (number of vertices in the graph).

The program was written in C++ using features up to C++17 and can be

compiled by any compiler that supports C++17 standard.

Chapter 1

Data structure

The data structure used to store the graph and all needed metadata is Doubly Connected Edge List (DCEL), a list of vertices, half-edges and faces connected by pointers. Each edge is represented by a pair of half-edges – directed edges having the same endpoints but opposite to each other.

The overview of the DCEL structure is as follows:
Half-edge:

- $from$ - a pointer to the vertex this half-edge is directed from (the *source* vertex).
- *to* - a pointer to the vertex this half-edge is directed to (the *target* vertex).
- opp - a pointer to the opposite half-edge.
- *next* - a pointer to the next half-edge with the same *source* vertex in the clockwise order.
- $left$ - a pointer to the face on the left side of this half-edge. Note that the pointer for the right side will be $left(opp)$.

Vertex:

- *to* - pointer to any half-edge directed from this vertex. All half-edges directed from this vertex form a connected list as *to*, $next(to)$, $next(next(to))$, ... in clockwise order.

Face:

- fst - pointer to any half-edge on the directed cycle bounding this face in counter-clockwise order. All these half-edges can be accessed as fst, $next(opp(fst))$, $next(opp(next(opp(fst))))$, ... in counter-clockwise order.

Note that those are only the attributes that were used to define the bare graph structure. Actual class definitions in the code contain many more attributes, most of which will be described later.

On the input however the program expects a simpler representation: for a graph G the vertices are presented with a set of numbers $[1 \dots |V(G)|]$ and the edges are stored in an array g of length $|V(G)|$ where $g[x]$ is the list of all edges incident with the vertex x sorted clockwise. Every edge appears exactly twice in this array in the form of two opposite directed edges (called *directions* from this point on) and the structure of the directions (called `simple_edge` in code) is as follows:

- *from* - the identifier of the source vertex of this direction.
- *to* - the identifier of the target vertex of this direction.
- *index* - an identifier of this direction. Both directions of the same edge have different indices.
- *link* - an identifier of the corresponding edge. Two directions of the same edge have the same link.

Importantly, the order in the list $g[x]$ is crucial, as with different orderings one could get different embeddings of the graph, possibly on different surfaces.

We use another structure to store the dual graph $G^\star$ (usually named in the code as H or G if the code is for dual graphs only). We define the connection between G and its dual by a function *dual* such that for $v \in V(G^\star)$ $dual(v)$ is defined as the corresponding face $f \in F(G)$ and $dual(f) = v$. Similarly, for each half-edge $e \in E(G)$ we define the connection to the dual as $dual(e) = e^\star \in E(G^\star)$ where the edge $e^\star$ of the dual graph is going from the face on the *left* side of e to the face on the *right* side of e. Also, we define $dual(e^\star) = e$. Note that there is no dual connection from $V(G)$ because the faces of the dual graph are irrelevant for our purposes and are not stored.

Lastly, for a face ϕ we define $|\phi| := |\{e \mid e \in E(G), left(e) = \phi\}|$, the length of the bounding closed walk of the face ϕ. Note that $|\phi| = \deg dual(\phi)$.

We will use a dual graph also as a flow network in the algorithm and to do that we add capacity and flow to all half-edges in the dual graph: $f(e^\star)$ and $c(e^\star)$ correspondingly.

From this point and until further notice DCEL representation of the graphs will be used. This means, in particular, that we view each edge $e \in E(G)$ as *directed* where $uv \in E(G) \iff vu \in E(G)$, i.e. each undirected edge is represented by a pair of directed half-edges.

Chapter 2

Nowhere-zero flows

One way to represent 3-colorings of a plane graph is by *nowhere-zero flows* in the dual graph. Let $col : V(G) \to \{0, 1, 2\}$ be a proper coloring of a plane graph G and let $G^\star$ be the dual graph of G. Note that for each edge $e \in E(G)$

$$col(to(e)) - col(from(e)) \equiv \pm 1 \pmod 3$$

Define $\delta(e) \in \{-1, +1\}$ so that

$$\delta(e) \equiv col(to(e)) - col(from(e)) \pmod 3$$

and set the flow of a dual half-edge $f(dual(e))$ as 1 if $\delta(e) = 1$, otherwise set the flow to 0. Now observe that $\delta(e) + \delta(opp(e)) = 0$, and thus

$$f(dual(e)) + f(dual(opp(e))) = 1$$

i.e. exactly one half-edge out of any pair of opposite half-edges in the dual graph will have flow 1.

Also define $\delta f(e^\star) \in \{-1, +1\}$ as $\delta f(e^\star) = f(e^\star) - f(opp(e^\star))$. Alternatively $\delta f(e^\star) = 2f(e^\star) - 1$, and thus we can deduce f from δf and vice versa. Note that $\delta f(e^\star) = \delta(dual(e^\star))$.

We call a flow f a *unit* flow if for every $e^\star \in E(G^\star)$ $f(e^\star) \in \{0, 1\}$. We call a unit flow a *nowhere-zero* flow if $\delta f(e^\star) = \pm 1$ for every $e^\star \in E(G^\star)$.

Note that our notion of a flow is slightly different from the usual one, as we allow the amount of flow leaving a vertex to be non-zero, i.e. a vertex can have a non-zero *excess*. Still, we can make some observation on the excess, which we denote as $\Delta f : V(G^\star) \to \mathbb{Z}$ and define as

$$\Delta f(v) = \sum_{e^\star \in E(G^\star) \,:\, to(e^\star) = v} f(e^\star) - \sum_{e^\star \in E(G^\star) \,:\, from(e^\star) = v} f(e^\star) \text{ for each } v \in V(G).$$

Note that for flows derived from 3-colorings as described above, the values of the excess are constrained as explained in the following lemma:

Lemma 2.1. *Let G be a plane graph, let $G^\star$ be the dual of G and let f be the flow in $G^\star$ defined by a 3-coloring of G as described above. Let ϕ be a face of G with the dual vertex v. It holds that $\Delta f(v) \in \{6k \mid k \in \mathbb{Z}\}$ for even $|\phi|$ and $\Delta f(v) \in \{3 + 6k \mid k \in \mathbb{Z}\}$ for odd $|\phi|$.*

Proof. Let C be the closed walk bounding ϕ. Without loss of generality suppose that the edges in C are directed clockwise around ϕ, so for each $e \in E(G)$, we have $left(opp(e)) = \phi$. This means that $to(dual(e)) = from(dual(opp(e))) = v$, i.e. the duals to half-edges in C are directed *to* v. Therefore

$$\Delta f(v) = \sum_{uv \in E(G^\star)} \delta f(uv) = \sum_{uv \in E(G^\star)} \delta(dual(uv)) = \sum_{e \in C} \delta(e).$$

By the definition of $\delta(e)$, we have $\sum_{e \in C} \delta(e) \equiv 0 \pmod 3$, and thus $\Delta f(v) \in \{3k \mid k \in \mathbb{Z}\}$.

To finish the proof it is only left to see that each edge in C contributes ± 1 to $\sum_{e \in C} \delta(e)$ and consequently to $\Delta f(v)$, and thus $\Delta f(v) \equiv |C| \pmod 2$. □

Conversely, as was first observed by Tutte [8], the existence of a flow with these properties implies that the graph is 3-colorable. Define a *boundary* on a graph $H = (V, E)$ as a function $b_H : V(H) \to \mathbb{Z}$, such that $\sum_{v \in V(H)} b_H(v) = 0$. We say that the boundary is *valid*, when for every $v \in V(H)$ it holds that $b_H(v) \in \{3+6k \mid k \in \mathbb{Z}\}$ for odd $\deg v$, $b_H(v) \in \{6k \mid k \in \mathbb{Z}\}$ for even $\deg v$ and $|b_{G^\star}(v)| \leq \deg v$. We will typically use this definition in the context $H = G^\star$. We will show that if there is a valid boundary for $G^\star$ such that there exists a unit flow with excesses equal to this boundary, then G is 3-colorable.

Theorem 2.2. *Let G be a plane graph, and let $G^\star$ be the dual of G. The graph G is 3-colorable iff there exists a valid boundary $b_{G^\star}$ and a unit flow f in $G^\star$ such that $b_{G^\star} = \Delta f$.*

Proof. The existence of a boundary with this property in 3-colorable graphs follows from Lemma 1 and its proof. The other implication can be proven by constructing a proper 3-coloring given a valid boundary $b_{G^\star}$ and a unit flow f whose excess matches this boundary.

First of all, we will show that there also is a nowhere-zero unit flow with the same excess. We will do this by transforming the flow f into a nowhere-zero flow. A pair of opposite half-edges uv and vu is called *directed* if $f(uv) \neq f(vu)$ and *undirected* otherwise. Our goal is to make every pair directed. Every undirected pair of opposite half-edges $\{uv, vu\}$ contributes 0 to the parity of $\Delta f(u)$ and $\Delta f(v)$ and every directed pair contributes ± 1. Note that for $v \in V(G^\star)$, $\Delta f(v) = b_{G^\star}(v) \equiv \deg v \pmod 2$, so for each dual vertex there is an even number of undirected pairs of half-edges incident to it.

Let G_E be the graph with $V(G_E) = V(G)$ such that for each undirected pair of half-edges $\{uv, vu\}$ in G there is an undirected edge uv in $E(G_E)$. From the statement above it follows that each component of G_E is an eulerian graph, and thus we can orient the edges in G_E so that $deg^+(v) = deg^-(v)$ for $v \in V(G_E)$. Now if an edge $uv \in E(G_E)$ is oriented from u to v, set the flow of the half-edge $uv \in E(G)$ to 1 and of the half-edge $vu \in E(G)$ to 0. From the properties of the orientation of G_E it is clear that this does not change the excess of f and causes all pairs of half-edges in G to be directed. Finally, since all pairs are directed, it is true that $f(opp(e)) \neq f(e)$ for every $e \in E(G^\star)$, and thus the flow is nowhere-zero.

Hence, we can assume that f is a nowhere-zero flow in $G^\star$ with $\Delta f(v) = b_{G^\star} \equiv 0 \pmod 3$ for every $v \in V(G^\star)$. Let T be a directed spanning tree of G with root in r and let T' be the reverse of T, i.e. $E(T') = \{opp(e) \mid e \in E(T)\}$. Set $col(r) = 0$ and color the vertices of G so that $\delta(e) = \delta f(dual(e))$ for each edge e of T; this can be done by setting the color of the vertices in order according to the distance from r. The color difference is always ± 1, so we get a proper coloring for T and T'. The only thing left to check is if the colors are different for edges in $E(G)/E(T \cup T')$. Let $e = uv$ be a half-edge from $E(G)/E(T \cup T')$. There is a path P_{vu} from v to u in $T \cup T'$ and it holds that $\sum_{e \in P_{vu}} \delta f(dual(e)) = \sum_{e \in P_{vu}} \delta(e) \equiv col(u) - col(v) \pmod 3$. Let C_{uv} be a cycle defined as $C_{uv} := \{uv\} \cup P_{vu}$. Note that in the plane graph cycles divide the plane into two parts, the dual of the cycle C_{uv} denoted as $C^\star_{uv}$ is a cut in $G^\star$, and consequently, the amount of flow going through $C^\star_{uv}$ is equal to the sum of $\Delta f(v)$ for the vertices on one side of the cycle. However $\Delta f(v) \equiv 0 \pmod 3$ for each $v \in V(G^\star)$ so the sum is also a multiple of 3, i.e. $\sum_{e \in C_{uv}} \delta f(dual(e)) = 0 \pmod 3$. Then

$$0 \equiv \sum_{e \in C_{uv}} \delta f(dual(e)) = \left(\sum_{e \in P_{vu}} \delta f(dual(e)) \right) + \delta f(dual(uv)) \pmod 3$$

$$0 \equiv col(u) - col(v) + \delta f(dual(uv)) \pmod 3$$

$$\delta f(dual(uv)) \equiv col(v) - col(u) \pmod 3$$

And since f is a nowhere-zero flow and $\delta f(e^\star) = \pm 1$ for every $e^\star \in E(G^\star)$, we have $col(u) \neq col(v)$. Therefore, col is a proper 3-coloring of G. □

By Theorem 2.2 we can decide whether G is 3-colorable by going over all valid boundaries, and for each valid boundary $b_{G^\star}$ testing whether exists a unit flow satisfying $\Delta f = b_{G^\star}$.

Iterating through all boundaries is done by a simple bruteforce search – for each vertex v in the dual graph the value of boundary is between $-\deg v$ and $\deg v$ and depending on the parity of $\deg v$ it belongs either to $\{6k | k \in \mathbb{Z}\}$ or to $\{3 + 6k | k \in \mathbb{Z}\}$. There can be exponentially many different boundaries for G;

however, for a face ϕ with $|\phi| = 4$, $b_{G^\star}(dual(\phi))$ can only be 0. Therefore, for *near-quadrangulations*, i.e. the graphs where the lengths of all faces are bounded by a constant and in which there are at most constantly many faces of length distinct from 4, there are constantly many choices of $b_{G^\star}$.

Once we chose a valid boundary function, we need to try to find a unit flow with the same excess. For every vertex $v \in V(G^\star)$, if $b_{G^\star}(v) < 0$, v has to generate exactly $|b_{G^\star}(v)|$ units of flow, if $b_{G^\star}(v) > 0$ the vertex has to consume $|b_{G^\star}(v)|$ units of flow, while every edge can transfer either 0 or 1 unit of flow. This is a classic problem of multi-source and multi-target maximum flow that can be solved by adding a *source* vertex s and a *target* vertex t and for all $v \in V(G^\star)$ if $b_{G^\star}(v)$ is positive, creating a half-edge from v to t with capacity $b_{G^\star}(v)$, otherwise creating a half-edge from s to v with capacity $-b_{G^\star}(v)$. For all other edges we set the capacity to 1. We then find the maximum flow from s to t. It is easy to see that the maximum flow will saturate all edges incident to the source and the target if and only if the boundaries of $b_{G^\star}$ are met in the corresponding flow in an unaugmented graph. We use Ford-Fulkerson algorithm to find the maximum flow. The asymptotic complexity of this algorithm is $O(|M||E|)$, where M is the size of the maximum flow. Note that M is bounded by $\sum_{v \in V(G^\star)} |b_{G^\star}(v)|$, which is constant for near-quadrangulations, so the maximum flow for one boundary choice will be found in linear time. Moreover, the number of different valid boundaries for a near-quadrangulation is constant, so the whole algorithm will run in linear time.

The implementation of this algorithm including comments can be found in `nquadcol.cpp`.

Chapter 3

Extending the precoloring of two cycles

As the next step, we consider the problem of extending a precoloring of two facial cycles C_1 and C_2 to the 3-coloring of the whole plane graph.

First, let us show how this problem can be solved in the case of one precolored facial cycle. A high-level description of the solution is as follows: We will find a nowhere-zero unit flow with the excess equal to a chosen boundary and additionally matching the color differences on the precolored cycle, and then construct a proper 3-coloring out of this flow using a slight modification of the algorithm from the proof of Theorem 2.2.

Let G be a DCEL representation of the graph, let ϕ be the face whose boundary is precolored, let $C \subseteq E(G)$ be the the set of all half-edges incident with ϕ and let $C^\star = \{dual(e) \mid e \in C\}$. Note that for $e \in C^\star$ the desired value of $\delta(dual(e))$ is determined by the precoloring of C. Hence, we fix the flow on edges of $C^\star$ so that $\delta f(e) = \delta(dual(e))$ and search for a nowhere-zero flow that matches it. In the following subsection we will describe a way to search for a nowhere-zero flow in $G^\star$ that extends the fixed flow in $C^\star$ or generally in any subgraph of $G^\star$.

Extending a nowhere-zero flow of a subgraph

Let us start by showing how to extend a nowhere-zero flow in a subgraph to a unit flow, an extension to a nowhere-zero flow will easily follow.

Theorem 3.1. *Let H be a graph, and let b_H be a boundary on H. Let S be a subgraph of H, let $H' := H - E(S)$ and let f_0 be a unit flow in S. For each $v \in V(H)$ let $b'(v) = b_H(v) - \Delta f_0(v)$. Then there exists a unit flow in H with boundary b_H and matching f_0 on S if and only if there exists a unit flow in H' with boundary b'.*

Proof. Let f_H be a unit flow in H, such that $\Delta f_H = b_H$ and $f_H(e) = f_0(e)$ for every $e \in E(S)$. Let $f_{H'}$ be a unit flow in H' with $f_{H'}(e) = f_H(e)$ for $e \in E(H')$. Then, for every $v \in V(H)$

$$\Delta f_{H'}(v) = \Delta f_H(v) - \Delta f_0(v) = b_H(v) - \Delta f_0(v)$$

and thus the excess of $f_{H'}$ is equal to b'.

Conversely, let $f_{H'}$ be a unit flow in H', such that $\Delta f_{H'} = b'$. We define a unit flow f_H as

$$f_H(e) := \begin{cases} f_0(e) & \text{if } e \in E(S) \\ f_{H'}(e) & \text{if } e \in E(H') \end{cases}$$

for $e \in H$. Note that for each $v \in V(H)$

$$\Delta f_H(v) = \Delta f_{H'}(v) + \Delta f_0(v) = b'(v) + \Delta f_0(v) = b_H(v)$$

□

Now suppose f_0 is a nowhere-zero flow and b_H is a valid boundary, then we can notice that $b'(v) \equiv \deg v \pmod 2$ and according to Theorem 2.2 there will be a nowhere-zero flow $f_{H'}$ with excess b', and thus f_H obtained from combining $f_{H'}$ and f_0 as shown above will also be a nowhere-zero flow.

Extending a precoloring of one cycle

Consider the case of Theorem 3.1 where H is the dual of a plane graph G and S is the dual of a facial cycle C in G bounding a face ϕ. Note that the edges in $C^\star$ are exactly the edges incident with $dual(\phi)$ in H, and thus $H' - dual(\phi) = H - dual(\phi)$.

We define *contracting* a face ϕ in a plane graph G (denoted as $G/\{\phi\}$) as an operation that corresponds to contracting every edge of G incident to ϕ. From the geometric perspective it can be seen as reducing the size of the face ϕ sufficiently and replacing it with a vertex. We call this new vertex a *representative* of the face ϕ in a new graph. Note that in our case the dual of $G/\{\phi\}$ is $H - dual(\phi)$.

Therefore to extend a fixed nowhere-zero flow f_0 in $C^\star$ to a nowhere-zero flow in G satisfying some $b_{G^\star}$ we need to:

- Construct $H' := G^\star - dual(\phi)$.
- Iterate through all valid boundary functions $b_{G^\star}$.
- For each plausible $b_{G^\star}$ create $b_H(v) := b_{G^\star}(v) - \Delta f_0(v)$.

- Search for a nowhere-zero unit flow in H' with excess equal to b_H.
- If there is a flow f' in H' with excess equal to *any* b_H, change f' into a nowhere-zero flow without changing the excess as shown in Theorem 2.2, transform b_H back to $b_{G^\star}$ and add vertex $dual(\phi)$ back with all fixed dual edges.

Suppose this procedure succeeds, and thus we have a nowhere-zero flow with excess $b_{G^\star}$ extending the fixed flow on $C^\star$. Now we can use this flow to construct a 3-coloring of G which will be an extension of the coloring of C. This can be done with the algorithm described in the proof of Theorem 2.2, except we choose the root r of T as one of the precolored vertices and start by giving it the prescribed color. The only thing left is to prove that the obtained coloring is indeed an extension of the precoloring of C.

Define the amount of flow f going through a path P as

$$\Phi_f(P) = \sum_{e^\star \in P^\star} \delta f(e^\star)$$

.

Lemma 3.2. *Let G be a plane graph, let $G^\star$ be its dual and let f be a nowhere-zero unit flow with excess equal to $b_{G^\star}$, where the boundary $b_{G^\star}$ is valid. Let v_1 and v_2 be vertices of G. For any two paths $P_1, P_2 \subseteq E(G)$ starting in v_1 and ending in v_2 it is true that $\Phi_f(P_1) = \Phi_f(P_2) \pmod 3$*

Proof. From Theorem 2.2 we know that there exists a 3-coloring col' of G such that for every edge in G, $\delta f(dual(e)) = \delta(e)$. In this case $\Phi_f(P_i) = \sum_{e^\star \in P_i^\star} \delta f(e^\star) = \sum_{e \in P_i} \delta(e) \equiv col'(v_2) - col'(v_1) \pmod 3$ for $i \in \{1, 2\}$. □

Theorem 3.3. *Let G be a plane graph and let H be a connected subgraph of G. Let $col' : V(H) \to \{0, 1, 2\}$ be a proper 3-coloring of H. Then col' can be extended to a 3-coloring of G if and only if there exists a valid boundary $b_{G^\star}$ and a nowhere-zero unit flow f with excess $b_{G^\star}$ such that $\delta f(dual(e)) = \delta(e)$ for every $e \in E(H)$ where $\delta(e)$ is deduced from the precoloring of H.*

Proof. Let T be a spanning tree of G. Choose a precolored vertex r as a root and color T so that $col(r) = col'(r)$ and $\delta(e) = \delta f(dual(e))$ for every $e \in E(T)$. In Theorem 2.2 it is shown that this is a proper 3-coloring of G.

For every precolored vertex v there exist paths $H_v \subseteq H$ and $P_v \subseteq T$ from r to v. From the preceding lemma $\Phi_f(H_v) = \Phi_f(P_v) \pmod 3$, and thus $col'(v) - col'(r) = col(v) - col(r)$. Since $col'(r) = col(r)$, it holds that $col'(v) = col(v)$ and the coloring col is indeed an extension of col'.

For the second implication, let col be an extension of col', define f as $\delta f(e) = \delta(dual(e))$ for each edge e of $G^\star$ and let $b_{G^\star} = \Delta f$. □

It is worth noting that this proof is valid for any connected precolored subgraph, not just a face boundary.

One can find the implementation of this part in `cycle_extension.cpp`. This part is not used in the algorithm.

Unfortunately, extending a precoloring of two facial cycles C_1 and C_2 is not as simple since the subgraph $C_1 \cup C_2$ is not connected and thus, even if we find a flow whose values on the edges of $C_1^\star \cup C_2^\star$ correspond to the precoloring, it may not be possible to turn this flow into a 3-coloring that matches the precoloring of both cycles. In order to solve this problem, it is thus necessary to prescribe the value of the flow over a path P connecting the two cycles.

Theorem 3.4. *Let G be a plane graph, let C_1 and C_2 be facial cycles of G, and let P be a path in G from a vertex $v_1 \in V(C_1)$ to a vertex $v_2 \in V(C_2)$. A precoloring of $C_1 \cup C_2$ can be extended to a proper 3-coloring of G if and only if there exists a valid boundary $b_{G^\star}$ and a nowhere-zero unit flow f with excess $b_{G^\star}$ in $G^\star$ matching the flows on $C_1^\star$ and $C_2^\star$ derived from col' and satisfying $\Phi_f(P) \equiv col'(v_2) - col'(v_1) \pmod 3$.*

Proof. Suppose first col is an extension of col'. Define f such that $\delta f(e) = \delta(dual(e))$ for every $e \in E(G^\star)$. It is easy to verify that f is a unit nowhere-zero flow satisfying the conditions of the theorem.

Conversely, suppose that f is a nowhere-zero unit flow satisfying the conditions described in the statement of the theorem. Then we can color the vertices of P, denoting this coloring as col_P, so that $col_P(v_1) = col'(v_1)$ and $\delta f(dual(e)) = \delta(e)$ for every edge e in P. Note that $col_P(v_2) = col'(v_2)$ from the last condition in the statement of the theorem, and thus, combined with the fact that $col_P(v_1) = col'(v_1)$, $col'(v)$ is equal to $col_P(v)$ for any other vertex $v \in V(P) \cap V(C_1 \cup C_2)$ as follows from Lemma 3.2. Let $H = C_1 \cup C_2 \cup P$ and note that H is a connected subgraph of G, as it includes C_1 and C_2 and a path between them. Extend col' by col_P. The conclusion then follows from Theorem 3.3. □

Our aim for the next subsection is to ensure that

$$\Phi_f(P) \equiv col'(v_2) - col'(v_1) \pmod 3$$

Therefore, we need an algorithm to determine whether there exists a nowhere-zero flow f' such that $\Phi_{f'}(P)$ is some given fixed value modulo 3 and $\Delta f = \Delta f'$.

Flows with the exact amount going through a path

An operation that we can apply on the flow without changing Δf is *inverting* a cycle. Let $C \subseteq G^\star$ be a cycle such that $f(e) = 1$ for every $e \in E(C)$ (we call such

cycles *flow-directed*). The new flow $f_\oplus$ obtained by *inverting* C in F is defined by

$$f_\oplus(e) := \begin{cases} f(opp(e)) & \{e, opp(e)\} \cap E(C) \neq \emptyset \\ f(e) & \text{otherwise} \end{cases}$$

We define $f \oplus C := f_\oplus$. We will show that this operation is sufficient for transitioning between any two flows with the same excess.

Theorem 3.5. *Let f_1 and f_2 be nowhere-zero flows in a graph H satisfying $\Delta f_1 = \Delta f_2$. Then there exist pairwise edge-disjoint flow-directed cycles $C_1, C_2, \dots, C_n \subseteq H$ such that $f_2 = f_1 \oplus C_1 \oplus C_2 \oplus \cdots \oplus C_n$.*

Proof. Consider the flow f_{avg} defined by $\delta f_{avg}(e) = \frac{\delta f_1(e) - \delta f_2(e)}{2}$ for every edge e of H. From the definition it follows that $\delta f_{avg}(e) = 0$ if and only if $\delta f_1(e) = \delta f_2(e)$. Now let

$$E_C = \{e \mid e \in E(H),\ \delta f_{avg}(e) = 1\}.$$

Note that $e \in E_C$ exactly when $e \in E(H)$ and $\delta f_1(e) = -\delta f_2(e) = 1$. Also it can be easily shown that for every $v \in V(H)$, $\Delta f_{avg}(v) = \frac{\Delta f_1(v) - \Delta f_2(v)}{2} = 0$, and thus there are as many half-edges in E_C entering v as ones leaving v. This implies that we can partition E_C into a set of pairwise edge-disjoint directed cycles $C_1, C_2, \dots, C_n$. Inverting each of the cycles gives us the flow $f_1 - 2f_{avg} = f_2$ as required. □

Corollary 3.6. *Let G be a plane graph, let $G^\star$ be its dual and let $b_{G^\star}$ be a boundary on $G^\star$. Let P be a path in G. Let $f_{\min}$ and $f_{\max}$ be nowhere-zero flows in $G^\star$ such that $b_{G^\star} = \Delta f_{\min} = \Delta f_{\max}$ and $\Phi_{f_{\min}}(P)$ is minimal and $\Phi_{f_{\max}}(P)$ is maximal among all nowhere-zero unit flows with this property. For every integer x there exists a nowhere-zero unit flow f_x such that $\Delta f_x = b_{G^\star}$ and $\Phi_{f_x}(P) = x$ if and only if $x \equiv |P| \pmod 2$ and $\Phi_{f_{\min}}(P) \leq x \leq \Phi_{f_{\max}}(P)$.*

Proof. It is easy to notice that if x does not satisfy the requirements in the statement, the flow f_x does not exist. To prove the equivalence we need to verify that, if x satisfies these requirements, f_x exists.

Before that, let us make a couple of observations:

Let $C \subseteq G^\star$ be a flow-directed cycle and let f be a nowhere-zero unit flow. Let $e_1, e_2, \dots, e_n$ be the edges of P that cross C in order along P. Then

$$\Phi_{f \oplus C}(P) - \Phi_f(P) = -2 \sum_{i=0}^{n} \delta f(dual(e_i)).$$

Note that C divides the plane into two parts S_1 and S_2 and whenever an edge $e_i \in E(P)$ crosses C, the endpoints of e_i lie in different parts. Therefore if e_i enters

(say) S_1, then e_{i+1} leaves S_1 and $\delta f(dual(e_i)) = -\delta f(dual(e_{i+1}))$ holds, since C is flow-directed. Thus

$$\Phi_{f\oplus C}(P) - \Phi_f(P) := \begin{cases} \pm 2 & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases}$$

holds. In other words, inverting C can change $\Phi_f(P)$ by at most 2.

From Theorem 3.5 it follows that there exist pairwise edge-disjoint flow-directed cycles $C_1, C_2, \dots C_n$, such that $f_{\min} \oplus C_1 \oplus C_2 \oplus \cdots \oplus C_n = f_{\max}$ and in a sequence $\Phi_{f_{\min}}(P), \Phi_{f_{\min}\oplus C_1}(P), \Phi_{f_{\min}\oplus C_1\oplus C_2}(P), \dots, \Phi_{f_{\max}}(P)$ the difference between every two neighboring elements is in $\{-2, 0, 2\}$ from the observation above. Therefore, we can choose one of these flows as f_x. □

However, this result cannot yet be applied directly to solve the problem of this chapter, since the resulting flow f_x might not match the flow derived from the precoloring of $C_1 \cup C_2$. To solve this we need to modify the graph so that it does not contain the edges with the prescribed flow.

Removing the precolored cycles

We will start with the following lemma:

Lemma 3.7. *Let G be a plane graph, let $G^\star$ be the dual of G, let ϕ_1 and ϕ_2 be faces in G, bounded by cycles C_1 and C_2 correspondingly and let P be a path in G between $v_1 \in V(C_1)$ and $v_2 \in V(C_2)$, such that $V(P) \cap V(C_1 \cup C_2) = \{v_1, v_2\}$. Let f be a nowhere-zero unit flow in $G^\star$. Let $G' := G/\{\phi_1\}/\{\phi_2\}$, with v_1 and v_2 being the representatives of ϕ_1 and ϕ_2 correspondingly and let $(G')^\star$ be the dual graph of G'. Finally, let f' be a nowhere-zero unit flow in $(G')^\star$, such that $f'(e) := f(e)$ for each $e \in E(G')$. Then P is a subgraph of G' and $\Phi_f(P) = \Phi_{f'}(P)$.*

Proof. The assumptions on P imply that no edge in P is incident to ϕ_1 or ϕ_2 and also the only vertices in P lying on $C_1 \cup C_2$ are v_1 and v_2 that will persist as the representatives of ϕ_1 and ϕ_2. This means that the contractions do not change P in any way. From the dual perspective, no edge in $P^\star$ is incident to $dual(\phi_1)$ or $dual(\phi_2)$, and thus, from the definition of f', the flow on $P^\star$ also does not change. □

In other words, if a path P touches the cycles C_1 and C_2 only in the endpoints, contracting ϕ_1 and ϕ_2 changes neither P nor $\Phi_f(P)$.

Theorem 3.8. *Let G, $G^\star$, ϕ_1, ϕ_2, C_1, C_2, P, G' be defined the same way as in the lemma above. Let col' be a proper precoloring of $C_1 \cup C_2$ and let f_0 be the unit flow*

derived from col′. Then col′ can be extended to a proper 3-coloring of G if and only if there exists a valid boundary b_G and a nowhere-zero unit flow f' in $(G')^\star$ with excess equal to $b_{G'} := b_{G^\star} - \Delta f_0$, such that $\Phi_{f'}(P) \equiv col'(v_2) - col'(v_1) \pmod 3$.

Proof. Note that $(G')^\star = G^\star - dual(\phi_1) - dual(\phi_2)$, and thus, from Theorem 2.2 it follows that the existence of the given f' is equivalent to the existence of a nowhere-zero unit flow f with excess equal to $b_{G^\star}$. Moreover, f can be constructed from f' and f_0, so that

$$f(e) := \begin{cases} f_0(e) & \text{if } e \in E(C_1^\star \cup C_2^\star) \\ f'(e) & \text{if } e \in E((G')^\star) \end{cases}$$

Therefore, using Lemma 3.7 we get that

$$\Phi_f(P) = \Phi_{f'}(P) \equiv col'(v_2) - col'(v_1) \pmod 3.$$

From here we can apply Theorem 3.4 on f which also gives an equivalence. □

With this we have transformed a problem of determining whether an extension of col' on G exists into a problem of finding a nowhere-zero flow with certain conditions in $(G')^\star$.

The last thing left to do is to be able to find $f_{\min}$ and $f_{\max}$:

Finding maximal and minimal flow on a path

Theorem 3.9. *Let f be a nowhere-zero flow in the dual of a plane graph G and let $P \subseteq E(G)$ be a directed path from v_1 to v_2. If there is a path $P^+ \subseteq E(G)$ from v_1 to v_2 with $\delta(dual(e)) = 1$ for every edge e of P^+, then $\Phi_f(P)$ is maximal possible among all nowhere-zero flows with the same excess.*

Proof. Let $f_{\max}$ be any nowhere-zero flow in $G^\star$ with $\Delta f_{\max} = \Delta f$. By Theorem 3.5, there are edge-disjoint directed cycles $C_1, C_2, \dots, C_n$ such that $f \oplus C_1 \oplus \cdots \oplus C_n = f_{\max}$. For contradiction suppose $\Phi_{f_{\max}}(P) > \Phi_f(P)$. This means there is a cycle C_i whose inversion increases $\Phi_f(P)$. This happens only if v_1 and v_2 lie on different sides of the C_i (i.e. one vertex lies outside of C_i, the other lies inside). But then the same is true for P^+ and inverting C_i should increase $\Phi_f(P^+)$ too, which is impossible. □

Therefore to get $f_{\max}$ we need to find a flow for which there is such a path P^+. From this, a subroutine for finding $f_{\max}$ follows:

1. Start with $T = \{v_1\}$.
2. If $v_2 \in T$, set $f_{\max} := f$ and stop.

3. If there is an edge $uv \in E(G)$, such that $u \in T,\ v \notin T$ and $\delta f(dual(uv)) = 1$, add v to T and go to step 2.

4. Otherwise let E be a cut between T and $E(G)/T$. Since E is a cut, $E^\star$ is an eulerian graph, decompose $E^\star$ into pairwise edge-disjoint cycles $C_1, C_2, \dots C_n$ and set $f := f \oplus C_1 \oplus \cdots \oplus C_n$. Go to step 2.

Throughout the run of the algorithm T contains vertices of G that are reachable from v_1 through the edges with $\delta f(dual(e)) = 1$. It is not difficult to verify that if G is connected, this algorithm will always finish. It is worth mentioning that in the step 4 there is no need to decompose $E^\star$ into cycles, inverting flow on every edge of $E^\star$ is enough. Overall, this algorithm can be implemented by a modified breadth-first search and will work in linear time if implemented carefully.

This way we will obtain $f_{\max}$. We can obtain $f_{\min}$ analogously.

Extending a precoloring of two cycles

Now we finally have enough ingredients to construct an algorithm that will extend a precoloring of two cycles.

First we define a subroutine $prescribed(G, b_{G^\star}, P, x)$ that for a graph G, a boundary $b_{G^\star}$ on $G^\star$ and a path P determines if a flow f_x exists such that $\Delta f_x = b_{G^\star}$ and $\Phi_{f_x}(P) \equiv x \pmod 3$:

- If a nowhere-zero unit flow with excess $b_{G^\star}$ does not exist, return "false".
- Otherwise let f be any such flow.
- Construct $f_{\min}$ and $f_{\max}$ from f by the algorithm above.
- Return "true" if there exists $y \in \mathbb{Z}$ such that

$$y \equiv |P| \pmod 2 \text{ and } \Phi_{f_{\min}}(P) \leq y \leq \Phi_{f_{\max}}(P) \text{ and } y \equiv x \pmod 3.$$

- Otherwise return "false".

The complexity of this subroutine depends on the complexity of finding $f_{\max}$ and $f_{\min}$, which is linear as shown above, and the complexity of finding a nowhere-zero unit flow with excess equal to a given boundary, which also has linear complexity for near-quandrangulations. Therefore, $prescribed(G, b_{G^\star}, P, x)$ runs in linear time when G is a near-quadrangulation.

Now we have everything prepared to formulate an algorithm to decide whether in a plane graph G, a precoloring col' of two facial cycles C_1 and C_2 that bound faces ϕ_1 and ϕ_2 can be extended to a 3-coloring of the whole graph G:

1. Create a graph G' from G by contracting ϕ_1 and ϕ_2, v_1 and v_2 being the representatives.
2. Find a path P from v_1 to v_2 in G', mark first and last edges of P as e_1 and e_2.
3. Map e_1 and e_2 back to G, remember the colors of $from(e_1)$ and $to(e_2)$ as col_1 and col_2.
4. Iterate through every valid boundary $b_{G^\star}$ for the graph G.
    - For the boundary $b_{G^\star}$ create the boundary $b_{G'}$ on $(G')^\star$ as defined in Theorem 3.8.
    - If $prescribed(G', b_{G'}, P, col_2 - col_1)$ is "true", return "true".
5. Otherwise return "false".

As we explained in the previous chapter, there are at most constantly many different valid boundaries for near-quadrangulations and for each chosen boundary we run a *prescribed* subroutine, which has a linear time complexity for near-quadrangulations. Thus, for a near-quadrangulation G and two precolored facial cycles C_1 and C_2 we can decide whether the precoloring can be extended into a proper 3-coloring of G in asymptotically linear time.

This algorithm was implemented in `double_cycle.cpp`.

Chapter 4

Short non-contractible cycles

The aim of the next two chapters will be extending the 3-coloring algorithm given in the previous chapters to the case of toroidal graphs. In this chapter we assume that the graph G is toroidal but not planar. The idea is to find a non-contractible cycle C in G and for every coloring of C split the cycle in two cycles with the same coloring and try to extend the coloring of these two cycles in the resulting planar graph by the routine described before. This approach will be explained later in more detail. To be able to try all colorings of C in linear time we need C to be short, the length should be constant in particular, and to make G planar after splitting the cycle C we need C to be non-contractible. Therefore, the task that we consider in this section is finding a short non-contractible cycle.

Non-contractible cycles in the dual graph

First let us describe a way to check if some cycle in G is non-contractible. For this we start by finding two non-contractible and homotopically non-equivalent cycles C_x and C_y in the dual graph $G^\star$ that we view as circling the torus by "latitude" and "longitude". This is done in the following way:

- Find a spanning tree $T \subseteq E(G)$
- Let E_c be $\{dual(e) \mid e \in E(G)/T\}$. Since G has a 2-cell embedding on a torus, the generalized Euler formula gives us $|E(G)| = |V(G)| + |F(G)|$, and we have

  $$|E_c| = |E(G)| - |E(T)| = |E(G)| - |V(G)| + 1 = |F(G)| + 1 = |V(G^\star)| + 1.$$

  Moreover, we can observe that the subgraph of $G^\star$ with edge set E_c is connected.
- Find a spanning tree $T^\star$ of G^* with $E(T^*) \subseteq E_c$, and let $E_c/E(T^\star) = \{e_x, e_y\}$.

- Let C_x be the unique cycle in $T^\star + e_x$.
- Let C_y be the unique cycle in $T^\star + e_y$.

Theorem 4.1 (Eppstein [9]). *Let G be a graph with a 2-cell embedding on a torus. Then the cycles C_x and C_y obtained as described above are non-contractible and homotopically non-equivalent.*

Note: this algorithm uses the conventional representation of undirected graphs where $E(G)$ and $E(G^\star)$ are sets of undirected edges. Let us direct C_x and C_y in any direction and proceed using the DCEL representation from this point.

Let $cross_x(e)$ for $e \in E(G)$ be 1 if $opp(dual(e)) \in C_x$, -1 if $dual(e) \in C_x$ and 0 otherwise. Let us remark that $opp(dual(e)) \in C_x$ intuitively means that e crosses C_x from left to right and $dual(e) \in C_x$ means e crosses C_x from right to left. We extend this definition to walks P in G by letting $cross_x(P) := \sum_{e \in P} cross_x(e)$. Analogously we define $cross_y$ for C_y. Given a directed cycle $C \subseteq G$, we can decide if C is contractible by counting $cross_x(C)$ and $cross_y(C)$.

Theorem 4.2. *Let G be a graph with a 2-cell embedding on a torus and let $C_x, C_y \subseteq G$ be non-contractible homotopically non-equivalent cycles. Then a cycle $C \subseteq G$ is contractible if and only if $cross_x(C) = cross_y(C) = 0$.*

Remark: Eppstein [9] shows this (for general surfaces) in the homology setting. We use the homotopy language to avoid having to introduce the homology formalism, noting that as is well-known, the fundamental group of the torus is commutative and thus coincides with the homology group.

Consequently, to find a shortest non-contractible cycle, it suffices to find a shortest cycle C_1 with non-zero $cross_x(C_1)$, a shortest cycle C_2 with non-zero $cross_y(C_2)$, and return the shorter of the two cycles.

Orientation of a graph

We define the *outdegree* of a vertex v in a directed graph $\vec{G}$ as a function

$$\deg^+_{\vec{G}} v := |\{(v,u) \mid (v,u) \in E(\vec{G})\}|$$

For a graph G, we define an *orientation* of G as a directed graph $\vec{G}$ such that $V(G) = V(\vec{G})$ and for each $uv \in E(G)$ exactly one of $\{uv, vu\}$ is contained in $E(\vec{G})$. Also, we call orientations of cycles *pseudocycles*, orientations of paths *pseudopaths* and orientations of walks *pseudowalks*.

Finally, if $\vec{G}$ is an orientation of a graph G and C is a subgraph of G, define $S_{\vec{G}}(C) = \vec{C}$ where $V(\vec{C}) = V(C)$ and $E(\vec{C}) = \{(u,v) \mid (u,v) \in E(\vec{G}) \wedge uv \in E(C)\}$, i.e. we get $\vec{C}$ by inducing the orientation of $\vec{G}$ on C.

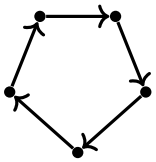


Figure 4.1 A cycle

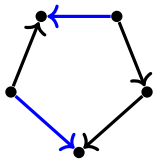


Figure 4.2 A pseudocycle

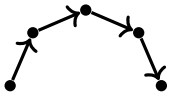


Figure 4.3 A path

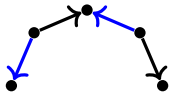


Figure 4.4 A pseudopath

Orientation with a bounded maximum outdegree

In this subsection we will define an algorithm that will return an orientation of every graph G and will show that for a special class of graphs, the maximum outdegree of the resulting orientation is bounded by a constant. This algorithm is particularly interesting for us because all graphs considered in the rest of this chapter have this property.

Define the *density* of a graph G as the value $\frac{|E(G)|}{|V(G)|}$. Note that the density of G is half of its average degree. Let M be the maximum of densities of all subgraphs of G. Since the density of G is at most M, there is a vertex $v_{\min} \in V(G)$ with $\deg_G v_{\min} \leq 2M$. Remove this vertex, orient $G - v_{\min}$ recursively, add $v_{\min}$ and leave only the half-edges directed out of $v_{\min}$. This way, we can obtain an orientation $\vec{G}$ of G with $\max_{v \in V(\vec{G})} \deg^+_{\vec{G}} v \leq 2M$. Let us call this algorithm a *constant-degree orientation algorithm* for the reasons explained in the following observation.

Observation 4.3. *Let $\mathscr{C}_k$ be a family of graphs, such that the density of every graph G in $\mathscr{C}_k$ is at most k and every subgraph of G also belongs to $\mathscr{C}_k$ (i.e. $\mathscr{C}_k$ is closed under the subgraph relation). Then for every graph in $\mathscr{C}_k$ the constant-degree orientation algorithm will return an orientation with the maximum outdegree bounded by $2k$.*

From this observation it follows that for every graph with a 2-cell embedding on a torus this algorithm will return an orientation with the maximum outdegree bounded by 6.

To be able to make use of this algorithm we will need the following observation:

Observation 4.4. *Let $\vec{G}$ be a directed graph with maximum outdegree M. Let v be a vertex in $V(\vec{G})$. Then there are at most M^n directed walks of length n in $\vec{G}$ starting in v.*

In particular, we can enumerate all directed cycles of constant length containing v in constant time. However, apart from the fact that we need to consider pseudocycles as well, we are interested only in non-contractible cycles, so we need to come up with a method for distinguishing them.

Differentiating non-contractible cycles

Let us now fix a constant ℓ, the maximum length of a non-contractible cycle in G that is of interest to us; in the program, $\ell = 5$. We keep track of the value of $cross_x$ using the following auxiliary graph:

We define the *ℓ-extension* of a graph $\vec{G}$ as the graph $\vec{H}$ with the vertex set being $V(\vec{G}) \times \{-\ell, \dots, \ell\}$ where for each edge $(u, v) \in E(\vec{G})$, $\vec{H}$ contains edges $((u, a), (v, b))$ for every $a, b \in \{-\ell, \ell\}$ such that $a + cr_x((u, v)) = b$.

Observation 4.5. *Let $\vec{G}$ be a graph with a 2-cell embedding on a torus and let $\vec{H}$ be an ℓ-extension of $\vec{G}$. We can embed $\vec{H}$ on a cylinder by cutting the torus embedding of $\vec{G}$ along C_x, thus getting a cylinder, and concatenating $2\ell + 1$ copies of this cylinder where the i-th cylinder contains the vertices of type (v, i) for $v \in V(\vec{G})$ and $i \in \{-\ell, \dots \ell\}$. Since a cylinder is homeomorphic to a subset of plane, $\vec{H}$ is a plane graph.*

Let G be a graph and let $\vec{G}$ be its orientation. Let $\vec{H}$ be an ℓ-extension of $\vec{G}$. Let $v = v_1$ be a vertex of $\vec{C}$, let $C = v_1 v_2 v_3 \dots v_n v_1$ be a cycle in G of size $n \leq \ell$ and let $\vec{C} = S_{\vec{G}}(C)$. Define the *unfolding* of $\vec{C}$ in v as the pseudopath $\vec{P} \subseteq \vec{H}$ of length n defined inductively with a sequence of pseudopaths $(\vec{P}_0, \vec{P}_1, \dots, \vec{P}_{n-1}, \vec{P}_n)$, where $\vec{P}_0 := \{(v, 0)\}$, $a_0 := 0$ and for each $i \in \{1, \dots n\}$

$$a_i := \begin{cases} a_{i-1} + cross_x(v_i v_{i+1}) & \text{if } v_i v_{i+1} \in E(\vec{C}) \\ a_{i-1} - cross_x(v_{i+1} v_i) & \text{if } v_{i+1} v_i \in E(\vec{C}) \end{cases}$$

$$\vec{P}_i := \begin{cases} \vec{P}_{i-1} + ((v_i, a_{i-1}), (v_{i+1}, a_i)) & \text{if } v_i v_{i+1} \in E(\vec{C}) \\ \vec{P}_{i-1} + ((v_{i+1}, a_i), (v_i, a_{i-1})) & \text{if } v_{i+1} v_i \in E(\vec{C}) \end{cases}$$

When defining $\vec{P}_n$, we let $v_{n+1} = v_1$. Finally, set $\vec{P} = \vec{P}_n$.

This operation can be viewed as cutting the cycle in v, mapping one end of the resulting path to $(v, 0)$, mapping the other end to $(v, cross_x(\vec{C}))$ and matching the rest of the vertices to the corresponding vertices of the extended graph.

Conversely, it clearly holds that any pseudopath $\vec{P}$ in $\vec{H}$ from $(v, 0)$ to (v, a) corresponds to a closed pseudowalk in $\vec{G}$ of the same length.

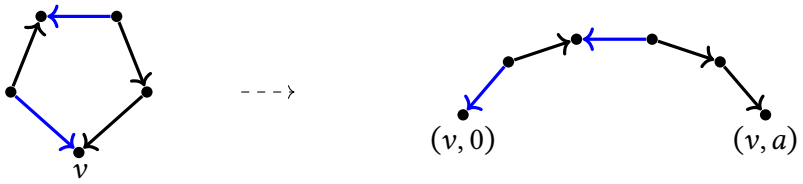


Figure 4.5 The unfolding of a pseudocycle

Theorem 4.6. *Let G be a graph with a 2-cell embedding on torus, let C_x be a non-contractible cycle in G and let $cross_x$ be defined the same way as before for C_x. Let $\vec{G}$ be an orientation of G obtained from a constant-degree orientation algorithm and let $\vec{H}$ be the ℓ-extension of $\vec{G}$. Suppose $\vec{P}$ is a shortest pseudopath in $\vec{H}$ such that P starts in $(v, 0)$ and ends in (v, a) for some vertex $v \in V(\vec{G})$ and $a \neq 0$. If the length of $\vec{P}$ is at most ℓ, then $\vec{P}$ is the unfolding of $S_{\vec{G}}(C)$ for a shortest cycle C in G such that $cross_x(C) \neq 0$.*

Proof. First, we can show that for every two distinct vertices of type (u, i) and (u, j) in $\vec{P}$ there is a path in $\vec{H}$ from $(u, 0)$ to $(u, i - j)$ and this pseudopath is shorter than $\vec{P}$ unless (u, i) and (u, j) are exactly the ends of $\vec{P}$. Therefore, there exists a *pseudocycle* $\vec{C} \subseteq \vec{G}$, such that the unfolding of $\vec{C}$ is $\vec{P}$.

Define $C \subseteq G$ as a cycle that is an orientation of $\vec{C}$. Note that $cross_x(C) \neq 0$. Let C' be a shortest (non-contractible) cycle in G such that $cross_x(C') \neq 0$ and let $\vec{C'} = S_{\vec{G}}(C')$, then $|cross_x(C')| \leq |E(\vec{C'})| = |E(C')| \leq |E(C)| \leq \ell$.

Let a pseudopath $\vec{P'} \subseteq \vec{H}$ be an unfolding of $\vec{C'}$ and let $(x, 0)$ and $(x, cross_x(C'))$ for some vertex $x \in V(\vec{G})$ be the start and the end of $\vec{P'}$ respectively. Recall that $cross_x(C') \neq 0$. Since $|E(C')| \leq |E(C)|$, we have $|E(\vec{P'})| \leq |E(\vec{P})|$ but $\vec{P}$ is a shortest pseudopath of this type, so $|E(C')| = |E(C)|$ and C is a shortest cycle in G. □

Consequently, the problem of this chapter can be transformed to a problem of finding a shortest pseudopath between certain vertices with length bounded by ℓ. Moreover, we will show that we do not need to consider *all* possible orientations of a path.

Iterating through unfoldings of all short pseudocycles

As we are interested only in case $\ell = 5$ we will concentrate on it from now on, as the details for the larger ℓ would be more involved.

For a pseudocycle $\vec{C}$, a vertex $v \in V(\vec{C})$ is a *source* if both edges incident to v are directed out of v. Let us now categorize all possible pseudocycles by the number of sources:

- Pseudocycles without any source are exactly directed cycles.
- Pseudocycles with one source have also exactly one sink vertex, and thus they can be obtained by joining the endpoints of two paths.

Figure 4.6 Examples of pseudocycles with one source

- There are only two pseudocycles with length at most 5 that contain two sources.

Figure 4.7 Pseudocycles with 2 sources

Now let G be a graph with a 2-cell embedding on a torus, let $\vec{G}$ be an orientation of G obtained by the constant-degree orientation algorithm and let $\vec{H}$ be the ℓ-extension of $\vec{G}$. Note that the maximum outdegree in $\vec{G}$ and, consequently, in $\vec{H}$, is bounded by a constant. Let us show which pseudopaths we need to consider in $\vec{H}$ to cover the unfoldings of all pseudocycles in $\vec{G}$.

The pseudocycles with no sources correspond to paths in $\vec{H}$. According to Observation 14, there are at most constantly many paths of length $\leq \ell$ starting in any vertex of $\vec{H}$. Thus, it would be asymptotically optimal to iterate through all vertices of G and for each vertex v iterate through all paths in $\vec{H}$ starting in $(v, 0)$ of length ≤ 5.

The pseudocycles with one source v can be unfolded from v and the unfolding corresponds to a union of two paths starting in $(v, 0)$ and (v, a) that meet in a vertex (Fig. 8). Thus, to iterate through all such unfoldings, for every pair of

vertices of type $(v, 0)$ and (v, a) for $v \in V(\vec{G})$ and $a \neq 0$ we iterate through all pairs of paths (P_1, P_2) of total length $\leq \ell$, P_1 starting in $(v, 0)$, P_2 starting in (v, a) and both ending in the same vertex. There are linearly many choices of (v, a) and for each pair $(v, 0)$ and (v, a) there are constantly many pairs of paths starting in these vertices. Therefore, iterating through pseudocycles with one source can be done in linear time.

Figure 4.8 An unfold of a pseudocycle with one source vertex

To iterate through unfoldings of pseudocycles with 2 source vertices we use the idea of augmentations first used by Kowalik and Kurowski [10]. For the graph $\vec{G}$; we define the graph G^+, such that $V(G^+) := V(\vec{G})$ and $E(G^+) := \{vw \mid (u, v), (u, w) \in E(\vec{G}), (u, v) \neq (u, w)\}$, we call this graph and edges in it *auxiliary*. Let us use the fact that $\vec{G}$ has an embedding on a torus and prove that the density of every subgraph of G^+ is bounded by a constant. From this fact it then follows, that the constant-degree orientation algorithm applied to G^+ will produce an orientation $\vec{G^+}$ with a maximum outdegree bounded by *some* constant and the graph $J := \vec{G} \cup \vec{G^+}$ will also have maximum outdegree bounded by a constant.

Theorem 4.7. *Let $\vec{G}$ be a directed graph with a 2-cell embedding on a torus and let $\Delta^+ := \max \deg^+_{\vec{G}} v$. Let G^+ be a graph, such that $V(G^+) = V(\vec{G})$ and $E(G^+) := \{vw \mid (u, v), (u, w) \in E(\vec{G}), (u, v) \neq (u, w)\}$. Then the density of every subgraph of G^+ is at most $\binom{\Delta^+}{2} + \frac{3\Delta^+}{2}$.*

Proof. Let F^+ be a subgraph of G^+, let G^- be the random graph obtained from $\vec{G}$ by randomly choosing an edge going out of v for each $v \in V(\vec{G}) \setminus V(F^+)$ and contracting the chosen edges. Let F^- be the subgraph of G^- induced on $V(F^+)$. Let vw be an edge in F^+, then there exists a vertex u, such that $\{(u, v), (u, w)\} \subseteq E(\vec{G})$. If u is outside of F^+, then the edge vw will appear in G^- if, for example, uw or uv gets contracted. Then the probability of vw being in G^- is at least $\frac{2}{\Delta^+}$. If u is inside of $\vec{F}$, then vw might never appear in G^-, but the amount of such edges is bounded by $\binom{\Delta^+}{2}|V(F^+)|$. Therefore, the expected number of edges of the random graph F^- is at least $\frac{2}{\Delta^+}\left(|E(F^+)| - \binom{\Delta^+}{2}|V(F^+)|\right)$

At the same time, F^- is a minor of a toroidal graph, and thus is a toroidal graph itself, so the number of edges in F^- is bounded by $3|V(F^-)|$, and so

$$\frac{2}{\Delta^+}\left(|E(F^+)| - \binom{\Delta^+}{2}|V(F^+)|\right) \leq 3|V(F^-)|$$

$$|E(F^+)| \leq \frac{3\Delta^+}{2}|V(F^+)| + \binom{\Delta^+}{2}|V(F^+)|$$

□

Note that for $\Delta^+ \leq 6$ the density of F^+ is at most 24.

Figure 4.9 Pseudocycles with edges from $\vec{G^+}$

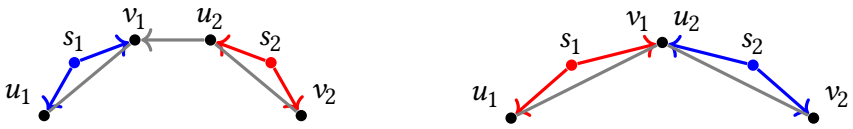


Figure 4.10 Pseudopaths with edges from $\vec{G^+}$

We now can use this theorem to iterate through pseudocycles with 2 sources effectively. Let G be a graph with a 2-cell embedding on torus and let $\vec{G}$ be the constant-degree orientation of G. Let G^+ be the auxiliary graph of $\vec{G}$ and let $\vec{G^+}$ be the constant-degree orientation of G^+. Let $J := \vec{G} \cup \vec{G^+}$. For every pseudocycle $\vec{C} \subseteq \vec{G}$ with sources s_1 and s_2 there is a pseudocycle $\vec{C^+} \subseteq J$ of length at most 3 with $V(\vec{C^+}) = V(\vec{C}) \setminus \{s_1, s_2\}$ (dashed and solid grey on Figure 9). Let $e := (u, v)$ be an edge in $\vec{G^+}$, then there should be a pair of edges (s, u) and (s, v) in $\vec{G}$. Then we define $origin(e) = \{(s, u), (s, v)\}$. There can be different pairs that e can originate from but we remember only one as our goal is to find only one shortest pseudocycle. Similarly, for a pseudocycle $\vec{C^+} \subseteq J$ we define the *original* pseudocycle $\vec{C}$ in $\vec{G}$ obtained by replacing all auxiliary edges in $\vec{C^+}$ by their origins. Note that we need to allow multi-edges because a pseudocycle of length 4 with two sources is represented by a cycle of length 2.

Note that the for every pseudocycle $\vec{C}$ of length at most 5 the pseudocycle $\vec{C^+}$ will have length at most 3. This means $\vec{C^+}$ can have at most one source and each pseudocycle with 2 sources in $\vec{G}$ is represented by a pseudocycle with at most one source in J. This leads us to the idea of finding shortest pseudopaths in J instead of $\vec{G}$. However, the auxiliary edges in J are not embedded and thus do not have $cross_x$ and $cross_y$ defined. Let $e := (u, v)$ be an auxiliary edge in J

and let $origin(e) = \{(s,u),(s,v)\}$. We define $cross_x(e) = cross_x((s,v)) - cross_x((s,u))$, similarly for $cross_y$. Let $\vec{H_J}$ be an ℓ-extension of J, we can see that for an auxiliary edge $e = (u,v)$ with the origin (s,u) and (s,v) and every integer a, such that $|a| \leq \ell$, $|a + cross_x((s,u))| \leq \ell$ and $|a + cross_x((s,v))| \leq \ell$ there is an edge from $(u, a + cross_x((s,u))$ to $(v, a + cross_x((s,v)))$ in $\vec{H_J}$. Then, if a pseudocycle $\vec{C^+}$ in J originates from a pseudocycle $\vec{C}$ in $\vec{G}$ of length at most 5, we can deduce the unfolding of $\vec{C^+}$ starting in some vertex v from an unfolding of $\vec{C}$ starting in the same vertex as shown in Figure 10. Therefore, every pseudocycle in J can be properly unfolded in $\vec{H_J}$.

The only issue left is that we need a shortest pseudopath in an ℓ-extension $\vec{H}$ of $\vec{G}$, which is $\vec{H_J}$ without the auxiliary edges, so we need to transform every unfolding found in $\vec{H_J}$ to the unfolding of the original pseudocycle in $\vec{H}$. For this we define a procedure $restore(P_J)$ that will return a corresponding pseudopath in $\vec{H}$ given a pseudopath in $\vec{H_J}$:

1. Set $\vec{P} := \emptyset$.
2. For every edge e in P_J:
   - If e is auxiliary, add $origin(e)$ to $\vec{P}$.
   - Otherwise add e to $\vec{P}$.
3. Return $\vec{P}$.

With this, we transform our problem of finding a shortest unfolding of a short non-contractible pseudocycle in $\vec{H}$ to iterating through all unfoldings of short pseudocycles with at most one source in $\vec{H_J}$.

The algorithm

Putting all ingredients together we are finally able to develop an algorithm that will find a shortest non-contractible cycle with length $\leq \ell = 5$ in a graph G with a 2-cell embedding on a torus:

1. Find C_x and C_y in G as shown in this chapter.
2. Compute $cross_x$ and $cross_y$ for every edge.
3. Construct $\vec{G}$ by orienting G with a constant-degree orientation algorithm.
4. Construct G^+ as shown above.

5. Construct $\vec{G^+}$ as a constant-degree orientation of G^+.
6. Construct $J := \vec{G} \cup \vec{G^+}$ and $\vec{H_J}$ as the ℓ-extension of J.
7. For every $v \in V(G)$ search through all paths in $\vec{H_J}$ starting in $(v, 0)$ of length $\leq \ell$.
    - If some path P ends in (v, a) for $a \neq 0$, update a current shortest pseudopath P_x with $\mathit{restore}(P)$.
8. For every $v \in V(G)$ and $a \neq 0$ search through all pairs of paths (P_1, P_2) in $\vec{H_J}$, P_1 starting in $(v, 0)$, P_2 starting in (v, a), both of length $\leq \ell$.
    - If for some P_1 and P_2 the ends coincide, update P_x with $\mathit{restore}(P_1 \cup P_2)$.
9. Create $\vec{H_J}$ as an ℓ-extension but using cross_y and repeat steps 3-8 storing the shortest path found in P_y.
10. Return the shortest of P_x and P_y.

The critical section of this algorithm is iterating through all paths of length at most ℓ starting in some vertex. It can be shown that in the chosen orientation there are constantly many such paths and thus this section takes constant amount of time for each vertex. Indeed, maximum outdegree of $\vec{H_J}$ is bounded by some constant M and the length of each path is at most ℓ, therefore the number of such paths is bounded by M^ℓ.

However, even though the number of such paths is bounded by a constant, in practice this number can be large and this is why one should proceed with extra care when implementing this part, especially the part with iterating through pairs of paths on step 7. For this reason a recursive procedure $\mathit{iterate}(P_1, P_2)$ is used for P_1 and P_2 being the intermediate paths:

1. Let s_1 be the first vertex of P_1 and let t_1 be the last vertex of P_1, define s_2, t_2 analogously for P_2.
2. If $|P_1| + |P_2| + 1 \geq 6$, return.
3. If there is an edge $t_1 s_2$ in J, update P_x with $\mathit{restore}(P_1 \cup \{t_1 s_2\} \cup P_2)$.
4. If there is an edge $t_2 s_1$ in J, update P_x with $\mathit{restore}(P_2 \cup \{t_2 s_1\} \cup P_1)$.
5. For every edge e_1 going out of t_1:
    - Call $\mathit{iterate}(P_1 \cup e_1, P_2)$.

6. For every edge e_2 going out of t_2:

    - Call $\mathit{iterate}(P_1, P_2 \cup e_2)$.

We can replace steps 3 and 4 in the main algorithm with $\mathit{iterate}(\{(v, 0)\}, \{(v, a)\})$ for $v \in V(G)$ and $a \neq 0$, since on step 3 of the procedure we update P_x with a path when P_2 has no edges. Step 2 introduces a way to cut the recursion branches that will not improve the solution. It is also possible to bound by the current minimum or, ultimately, stop the search once we found a non-contractible 4-cycle, which would significantly decrease average execution time, however this is a heuristic and we did not include it for more precise benchmarks. Another optimization lies in steps 3 and 4, where the search for the fitting edge can be done with associative dictionaries based on, for example, hashing. This breaks linear time complexity of the algorithm but greatly speeds up the implementation.

Other than that, constant-degree orientation and ℓ-extension can be implemented in linear time and it is clear that we conduct a path search at most twice in every vertex of $\vec{H}_J$, therefore this whole algorithm works in linear time.

This algorithm was implemented in `short_cycle.cpp` and the algorithm for finding C_x and C_y was implemented in `non_cont_cycles.cpp`.

Chapter 5

3-coloring for toroidal graphs

The previous chapters gave us enough background to build the 3-coloring algorithm for near-quadrangulations of the torus containing a noncontractible cycle of length at most 5. Let G be a graph with an embedding on a torus. We will assume that the embedding is 2-cell, if it is not, then the graph is a plane graph and we can use the approach from the chapter with the 3-coloring algorithm for plane graphs. This is simply verifiable by checking the characteristic number of the embedding: an embedding of a connected graph G on a torus is not 2-cell exactly if $|V(G)| - |E(G)| + |F(G)| = 2$.

Now the approach is to convert G into a plane graph and use already developed techniques to determine whether a 3-coloring exists.

This approach only works if G contains a short non-contractible cycle. We have an algorithm that finds such a cycle efficiently if its length is at most $\ell = 5$, and this turns out to be sufficient for our purposes due to the following result.

Theorem 5.1 ([6]). *Let G be a triangle-free graph with a 2-cell embedding on a torus. If G contains no non-contractible cycle of length at most 5, then G is 3-colorable.*

Hence, we can assume G contains a non-contractible cycle C of length at most five, which we found using the algorithm from the previous section.

Next, we *cut* the graph G by C: Denote one side of C as the left side and the other as the right side. Split C into two cycles of the same length named C_L and C_R and connect the edges incident to $V(C)$ from the left side to $V(C_L)$ and the rest of them to $V(C_R)$.

After cutting the torus along a non-contractible cycle C the resulting surface will be a cylinder, which is homeomorphic to a subset of the plane, and thus after this transformation G will be planar.

Now we will use the fact that the length of C is bounded by a constant:

Observation 5.2. *Let G be a graph with a 2-cell embedding on a torus, and let $C \subseteq G$ be a non-contractible cycle of length bounded by a constant. Let H be*

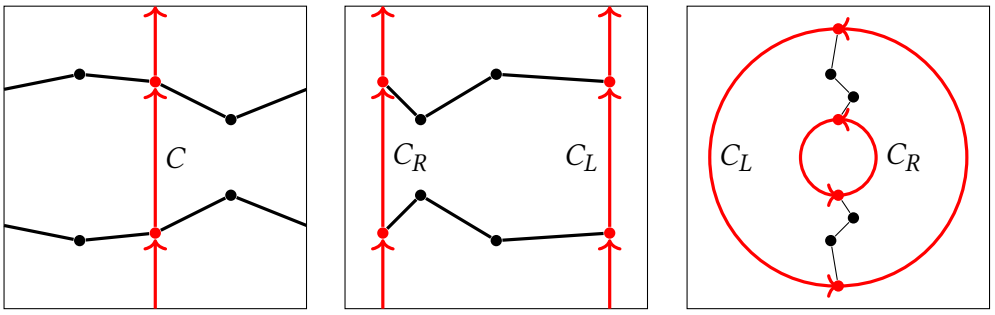


Figure 5.1 Cutting along a non-contractible cycle.

the graph obtained by cutting G along C where C was split to C_L and C_R. Let col' : $V(C) \to \{0,1,2\}$ be the precoloring of C and let col_H : $V(C_L \cup C_R) \to \{0,1,2\}$ be a precoloring, obtained by applying col' on C_L and C_R. Then col' can be extended to a 3-coloring of G if and only if col_H can be extended to a 3-coloring of H.

From this an algorithm $threecol_4(G)$ for determining the existence of a three-coloring of a triangle-free graph G with a 2-cell embedding on a torus follows:

1. Find a shortest non-contractible cycle in G with length ≤ 5.
2. If there is no such cycle, G is 3-colorable, return "true".
3. Otherwise let C be a shortest non-contractible cycle in G.
4. Create H from G by cutting G along C where C is split into cycles C_L and C_R.
5. Iterate through all proper precolorings of C.
    - For each such precoloring col' let col_H be col', applied on C_L and C_R. Verify whether col_H can be extended to a 3-coloring of H using the cycle extension algorithm from the corresponding chapter. If it can, then H is 3-colorable, and so is G. Return "true".
    - If no such extendable precoloring exists, return "false".

As shown in the previous chapter, step 1 is done in linear asymptotic time. If implemented carefully, cutting a graph along a cycle of constant length takes asymptotically linear time. Since the size of C is bounded by a constant, the number of valid precolorings is also constant, therefore we will need to verify whether the precoloring is extendable for a constant number of precolorings. However, we cannot guarantee that the verification will take linear asymptotic time, only in case of H being a near-quadrangulation. Note that if G is a near-quadrangulation, then H is a near-quadrangulation as well. Thus, if G is a near-quadrangulation, the last step will also have linear time complexity.

From this it follows, that this algorithm will decide if G is 3-colorable and will do so in asymptotically linear time if G is a near-quadrangulation.

Chapter 6

Eliminating non-facial 4-cycles

Motivation

In the previous chapter we finished an algorithm that will decide if a graph embedded on a torus is 3-colorable. That algorithm has linear time complexity for near-quadrangulations. Note that we can safely assume that the given graph is not planar, since we work only with triangle-free graphs and planar triangle-free graphs are 3-colorable by Grötzsch's theorem.

Naturally, the next thing to do is to try and transform a more general graph into a near-quadrangulation and apply the algorithm we have. Essentially, this near-quadrangulation should be 3-colorable exactly when the original graph is. For this reasons the following operation is offered:

Let G be a graph with a 2-cell embedding on a surface Σ where Σ is not the sphere, and let $K \subseteq G$ be a contractible cycle. Then K divides S into two parts, exactly one of them is homeomorphic to a disk, let this part be the *interior* of K.

Let G be a graph with a 2-cell embedding on a torus, and let $K \subseteq G$ be a contractible cycle of length 4 (from now on a *4-cycle*), We let $G \odot K$ denote the subgraph of G obtained by *eliminating the interior* of K in G, that is removing all edges and vertices drawn in the interior of K.

The following stronger form of the Grötzsch's theorem was proved by Aksionov [11]:

Theorem 6.1. *Let G be a triangle-free plane graph with an outer face bounded by a 4-cycle K and let col_K be a 3-coloring of K. Then the col_K can be extended to a 3-coloring of G.*

Corollary 6.2. *Let G be a triangle-free graph with a 2-cell embedding on a torus, and let K be a contractible 4-cycle in G. Then G is 3-colorable if and only if $G \odot K$ is colorable.*

To motivate the usefulness of this operation, let us note the following result of Dvořák and Pekárek [6]:

Theorem 6.3. *Let G be a triangle-free graph with a 2-cell embedding on torus, and assume G contains no non-contractible 4-cycles. If G is not 3-colorable, then there exists a subgraph $H \subseteq G$ that is a quadriangulation and is not 3-colorable.*

In this chapter we will design an algorithm that will determine the existence of 3-coloring in asymptotically linear time for triangle-free graphs embedded on a torus with no non-contractible 4-cycles. The last limitation will be resolved in the next chapter.

The idea of the algorithm is to iteratively eliminate the interiors of non-facial 4-cycles in G until all 4-cycles in G are facial (we call this process *compression*). Compression is a finite process, because every non-facial cycle contains at least 2 faces in the interior, so each interior elimination reduces the number of faces.

If after the compression we end up with a quadrangulation H, check its 3-colorability. If H is not 3-colorable, G is not 3-colorable as well. If H is 3-colorable, we can extend the coloring of H to a coloring of G according to Corollary 7.2. The only case left is when the compressed graph is not a quadrangulation. Then we claim the following:

Theorem 6.4. *Let H be a triangle-free graph with a 2-cell embedding on a torus Σ with no non-contractible 4-cycles. If H contains no contractible non-facial 4-cycle as a subgraph and H is not a quadrangulation, then H is 3-colorable.*

Proof. Let ϕ be a face of H with $|\phi| > 4$ and let the cycle bounding ϕ be C. For contradiction, assume that H is not 3-colorable, then by Theorem 7.3 there is a quadrangulation $Q \subset H$ that is not 3-colorable. Say ϕ is contained inside of some face $\phi_Q \in F(Q)$, denote the cycle bounding ϕ_Q as C_Q. Since C_Q bounds a face of a quadrangulation, it is contractible We will show that C_Q is a non-facial cycle in H, which would also lead to a contradiction.

Since $|C| > |C_Q|$, C_Q can not contain all vertices of C, so at least one vertex of C is contained in ϕ_Q, and this is not a face in H. The other side of C_Q can not be a face in H, either: since ϕ_Q is homeomorphic to a disk, Σ/ϕ_Q is not homeomorphic to a disk and the embedding of H is 2-cell. Consequently, the existence of such quadrangulation $Q \subseteq H$ would contradict the fact H has no non-facial 4-cycles, and thus H is 3-colorable according to Theorem 7.3. □

From this theorem it follows that if the subgraph $H \subseteq G$ obtained by compression G is not a quadrangulation, then G is 3-colorable due to Corollary 7.2 and Theorem 7.4.

Therefore, the strategy is to compress G and to obtain a graph $H \subseteq G$ as the result of the compression. If H is not a quadrangulation, G is 3-colorable, otherwise

determine 3-colorability of H by the algorithm from the previous chapter. Note that that algorithm works in linear time for quadrangulations.

Let us first show how to find and eliminate the interior of a contractible 4-cycle, later we will develop an algorithm that will iterate through all contractible 4-cycles in G and eliminate their interiors using procedures defined in the following sections.

Determining the interior

Once we have a contractible 4-cycle $K \subseteq G$ in order to eliminate the interior we need to first determine which side of K is the interior. For this we will extend the notion of an ℓ-extension:

Let G be a graph with a 2-cell embedding on a torus, let C_x and C_y be non-contractible homotopically inequivalent cycles in the dual of G, and let $cross_x$ and $cross_y$ be the crossing functions of C_x and C_y. Then the *(ℓ_x,ℓ_y)-extension* of G is the graph H, such that $V(H) = V(G)\times\{-\ell_x, \dots, \ell_x\}\times\{-\ell_y, \dots, \ell_y\}$ and for a_x, a_y, b_x, b_y, u, v there is an edge H from (u, a_x, a_y) to (v, b_x, b_y) exactly if there is an edge uv in G, $b_x = a_x + cross_x(uv)$ and $b_y = a_y + cross_y(uv)$.

Let us visualize this transformation, see Figures 11 and 12. Suppose for simplicity that the cycles C_x and C_y intersect in exactly one vertex (this does not have to be the case, complicating the visualizations but not affecting the validity of the results). The (ℓ_x,ℓ_y)-extension is done by cutting the torus along C_x and C_y, thus getting a disk, inside which a graph is embedded, except some of the edges (specifically, the edges with non-zero $cross_x$ or $cross_y$) are cut into two.

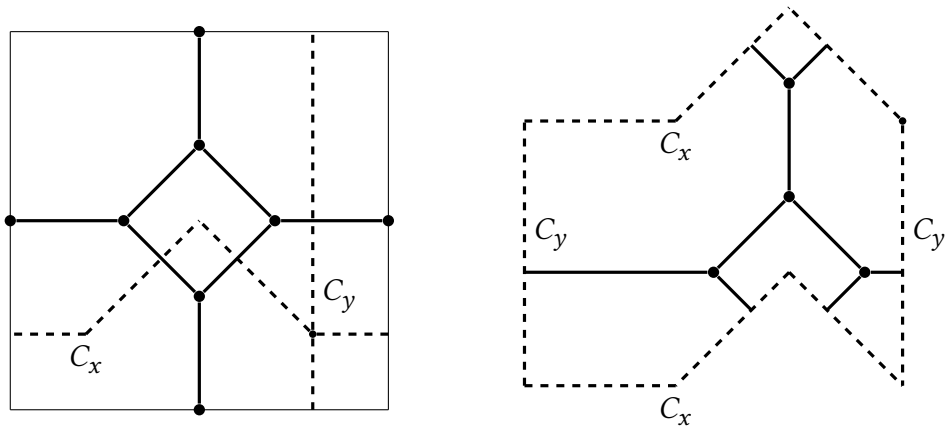


Figure 6.1 Cutting a torus into a disk

Next, glue $\ell_x \cdot \ell_y$ copies of this disc into a grid $\ell_x \times \ell_y$ by connecting the opposite sides of C_x horizontally and analogously for C_y vertically, connecting

the corresponding parts of cut edges. In the end some of the edges might still be cut into two, remove them. As a result we get the embedding of H in the plane.

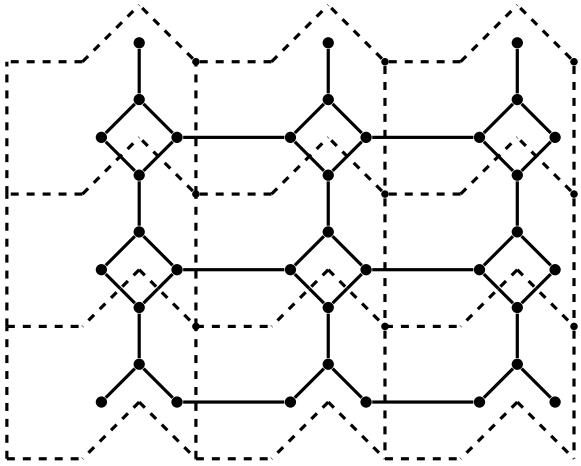


Figure 6.2 (1,1)-extension of a graph

Similarly, we define the *unfolding* of a cycle: let G be a graph with a 2-cell embedding on a torus, let H be the (ℓ_x,ℓ_y)-extension of G. The unfolding of a cycle $K = v_1 v_2 \dots v_n$ is a walk $\vec{K} = \vec{v}_1\vec{v}_2 \dots \vec{v}_{n+1}$ in H, such that

$$\vec{v}_i := (v_i, \sum_{k=2}^{i} cross_x(v_{k-1}, v_k), \sum_{k=2}^{i} cross_y(v_{k-1}, v_k))$$

To accommodate an unfolding of a cycle of length four a $(4, 4)$-extension will clearly be enough. However, we can note that the unfolding of a contractible cycle is a cycle as well, therefore the last vertex should be of form $(v_1, 0, 0)$ and the unfolding will fit in a (2,2)-extension. Moreover, if we allow the cycle to start in (v_1, x, y) for any x and y, we can fit the unfolding of every contractible 4-cycle in a (1,1)-extension.

A contractible cycle K divides the torus into a disk and the rest of the torus. The unfolding $\vec{K} \subseteq H$ of K also divides the plane into a disk and the unbounded part, where the disk in H bounded by $\vec{K}$ contains the same subgraph as the disk in G bounded by K, the disks being the interiors of the corresponding cycles. Furthermore, since K and $\vec{K}$ are oriented in the same direction, K is oriented counterclockwise relatively to the interior if and only if the same is true for $\vec{K}$. This means, to find the interior of K in G we can find the interior of $\vec{K}$ in a plane graph H.

Determining the interior in the plane graph

We have translated the problem of determining the interior of a cycle from toroidal graphs to plane graphs. We know that there is exactly one unbounded face in any plane graph, that is the outer face. Since every interior is bounded, the interior of a cycle C can not contain the outer face and this uniquely determines the side of C that bounds the interior.

The high-level strategy is to assume that C is directed counterclockwise around the interior, then we compute the number of faces inside the interior of C in a way that depends on the made assumption; if we get obviously wrong results, we can conclude that C is directed around the interior in the clockwise manner, otherwise the assumption was correct.

Theorem 6.5. *Let H be a plane graph with outer face ϕ, let C be a cycle in G, directed counterclockwise around its interior, and let S be the interior of C. Let $H^\star$ be the dual of H, and let T be a directed spanning tree of $H^\star$ rooted in $dual(\phi)$ with all edges directed from the root. Let $sz : E(T) \to \mathbb{Z}^+$ be the function, assigning to each edge (u, v) in T the number of vertices in the subtree of T rooted in v. Then the number of faces lying in S is*

$$\sum_{e \in E(T)\ :\ dual(opp(e)) \in C} sz(e) - \sum_{e \in E(T)\ :\ dual(e) \in C} sz(e).$$

Proof. Let v be a vertex in $H^\star$ and let P be the path in T from $dual(\phi)$ to v. Since ϕ does not lie in S, the number of times P enters S is the same as the number of times P leaves S if v does not lie in S, otherwise P enters one more time. Let $\chi(v) = |\{e \mid e \in E(P), e \text{ enters } S\} - |\{e \mid e \in E(P), e \text{ leaves } S\}$. It will be 1 if v lies in S, 0 otherwise. Consequently, we will get the number of dual vertices (and thus, also faces) inside S as $\sum_{v \in V(H^\star)} \chi(v)$. Notice that every edge e in T entering S contributes $sz(e)$ to the sum and every e in T leaving S contributes $-sz(e)$. Therefore the amount of faces in $H^\star$ inside of S is equal to

$$\sum_{e \in E(T) \text{ enters } S} sz(e) - \sum_{e \in E(T) \text{ leaves } S} sz(e).$$

To end the proof we use the assumption on the orientation of C and note that $e \in E(H^\star)$ enters C exactly if $dual(opp(e))$ is in C and leaves S if $dual(e)$ is in C. □

Note that the result of the proven expression has the opposite sign if C is directed the opposite way and that there is at least one face inside S. Then C is counterclockwise relatively to S exactly if the result of the expression above is positive. Another important remark is that the construction needed to find the interior does not need to change even if the graph G is changed after eliminating

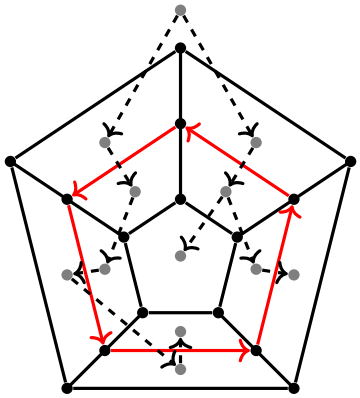


Figure 6.3 The construction with a spanning tree of the dual

a cycle: The interior of a cycle in a subgraph of G is the same as the interior of this cycle in G.

We will now define a procedure $interior(C, H, T, sz)$ that will orient a given contractible pseudocycle C counterclockwise around its interior in a plane graph H, where T is a directed spanning tree of the dual of H rooted in the outer face of H and sz is the function defined in the previous part of the chapter.

1. Orient all edges of C to one direction, no matter which one.
2. Initialize S to be 0.
3. For every edge e of C, if $dual(opp(e)) \in T$, add $sz(dual(opp(e)))$ to S
4. For every edge e of C, if $dual(e) \in T$, subtract $sz(dual(e))$ from S
5. If $S < 0$, reverse all edges of C
6. Return C

We can clearly see that for a pseudocycle of constant length this algorithm runs in constant asymptotic time. To find the interior of a contractible cycle in a triangle-free graph G with a 2-cell embedding on a torus we can use $interior(C, H, T, sz)$ with H being the (2,2)-extension of G. Since at any point of time we do not change the embedding of G but only delete vertices and edges, H can be static and we can compute H, T and sz before the first invocation and reuse the same structures in every call of $interior(C, H, T, Z)$.

Eliminating the interior of a 4-cycle

Once we have determined the direction in which the interior is, we need to remove everything inside of it. Let G be a graph with a 2-cell embedding on torus and let C be a contractible cycle in G, oriented counter-clockwise around its interior. We define procedure *eliminate*(C, G) that removes all edges, vertices and faces in the interior of C:

1. Let Q be a queue of faces, Push $left(e)$ for every edge e in C to Q.
2. Until Q is empty:
   - Pop a face F out of Q.
   - Mark F as pending deletion.
   - For every edge e adjacent to F, such that neither e nor $opp(e)$ belong to C, mark e and $opp(e)$ as pending deletion and add $left(opp(e))$ to Q if it has not been there yet.
3. Delete all faces and edges pending deletion, delete vertices with all incident edges deleted.
4. Add one face F_n and set its boundary to C (more specifically, update *next* for all edges from C and set one of the edges as $first(F_n)$).

The time complexity of *eliminate*(C, G) is linear to the number of faces in the interior of C, but every face accessed during the elimination is deleted and only one new face is created. Therefore the amortized complexity of *eliminate*(C, G) is $O(|F(G)| + N_C)$, where N_C is the number of invocations of *eliminate*(C, G). Since invocation for a non-facial 4-cycle decreases the number of faces, we have $N_C \in O(|F(G)|)$. Therefore, the total time complexity of all calls of *eliminate*(C, G) will be linear to the size of G.

Compression

In this section we will define the compression algorithm.

To compress a graph G one needs to iterate through all its non-facial contractible 4-cycles. To do this we will mostly use techniques and tools developed in Chapter 5.

Let G be a graph with a 2-cell embedding on a torus and let $\vec{G}$ be an orientation of G obtained by applying the constant-degree orientation algorithm on G. Let $\Delta^+ := \max_{v \in V(G)} \deg^+_{\vec{G}} v$. Then, the same as in the chapter with short non-contractible cycles we will consider all possible cases of orientations of 4-cycles

in $\vec{G}$, iterate through the pseudocycles of length 4 in $\vec{G}$, translate each pseudocycle to a cycle in G and eliminate the interior of every such cycle.

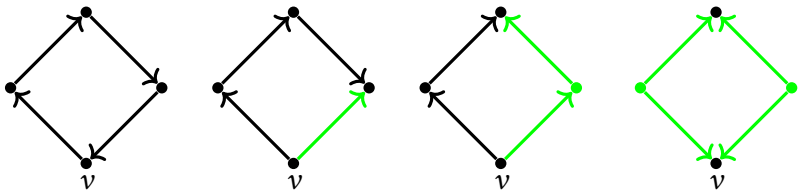


Figure 6.4 Orientations of a 4-cycle

Again, we categorize the pseudocycles of length 4 by the number of sources:

- No source: We can iterate through all such pseudocycles of length 4 containing a vertex v by iterating through all walks of length 4 starting in v, the number of which is bounded by $(\Delta^+)^4$.
- One source: we can iterate through all pseudocycles of length 4 with the only source being v by iterating through all pairs of paths starting in v with the total length of the paths equal to 4. The number of such pairs is bounded by $3(\Delta^+)^4$
- Two sources: Iterating through pseudocycles of length 4 with two sources can be done in the following way: Let v be a vertex in $\vec{G}$ and let $(u_1, v), (u_2, v), \dots (u_k, v)$ be all edges incident to v and directed towards v. For each $i \in \{1, \dots, k\}$ iterate through all edges going out of u_i and if there are $i \neq j$, such that there exist edges (u_i, a) and (u_j, a) for $a \neq v$, then there exists a pseudocycle of length 4 with vertices v, u_i, a, u_j.

Let us elaborate on the iteration algorithm from the last bullet. For vertices v and a in $\vec{G}$, let $in_v(a) = \{x \mid (x, a), (x, v) \in E(\vec{G})\}$. For every vertex $a \in V(\vec{G})$ iterate through all pairs $x, y \in in_v(a)$ and for every such pair a pseudocycle with vertices v, x, a, y exists with x and y being the sources (note that since G has no cycles of length ≤ 3, we can specify a 4-cycle with four vertices unambiguously). With this we can define a prototype algorithm for iterating through all 4-cycles with 2 sources:

- For every vertex v in $V(\vec{G})$:
  1. For every edge $e := (u_i, v)$ entering v from $E(\vec{G})$:
     - For every $(u_i, a) \in E(\vec{G}), a \neq v$ add u_i to $in_v(a)$.
  2. For every vertex a ,where $in_v(a)$ is non-empty:

- For every pair $x, y \in in_v(a), x \neq y$ eliminate the interior of a cycle with vertices a, x, v, y.

3. Clear in_v.

Of course, there can be a vertex a, for which the size of $in_v(a)$ after step 1 might be up to $\Theta(|V(G)|)$, so this solution is not effective enough. However, we can bound the size if we eliminate the cycles at the same time while calculating in_v:

- For every vertex v in $V(\vec{G})$

1. For every edge $e := (u_i, v)$ entering v from $E(\vec{G})$.
   - For every $(u_i, a) \in E(\vec{G}), a \neq v$
     - Add u_i to $in_v(a)$.
     - For every pair $x, y \in in_v(a), x \neq y$ eliminate the interior of a cycle with vertices a, x, v, y; the deleted vertices are also removed from all the sets $in_v(a)$ that contain them.
2. Clear in_v.

The fact that this improves the complexity is not evident but with the help of the following observation it can be shown that at any state of the algorithm $|in_v(a)| \leq 3$ holds for every v and a.

Observation 6.6. *Let S be a graph, embedded on a torus, such that $V(S) = \{u, v, x_1, x_2, x_3\}$ and $E(S) = \{\{u, x_i\} \mid i \in \{1, 2, 3\}\} \cup \{\{x_i, v\} \mid i \in \{1, 2, 3\}\}$. If the embedding of S contains no non-contractible 4-cycles,then S contains a 4-cycle whose interior is not a face.*

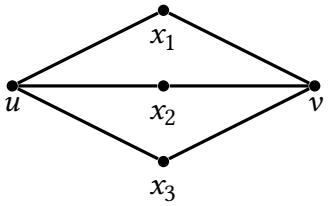


Figure 6.5 Graph S

The idea is that once we add the third vertex to $in_v(a)$, from 7.6 it follows that there should be a cycle $C := vx'ay'$ in G with $x', y' \in in_v(a)$, such that there is $z \in in_v(a) \setminus \{x', y'\}$ that lies in the interior of C. By eliminating the interiors of $vxay$ for all pairs $x, y \in in_v(a)$ in the next step we will eventually eliminate the

interior of C, thus deleting z and decreasing the size of $in_v(a)$ back to 2. With this the size of $in_v(a)$ for any v and a can not be more than 3.

Consequently, at each iteration of u_i a constant amount of new 4-cycles is found. Taking the amortized complexity of *eliminate* into account, we can see that the time complexity of this routine is linear to the size of G.

This concludes the case of pseudocycles with two sources. Apart from that, we need to precompute the augmenting structures for determining the interior and eliminate the interiors of contractible pseudocycles of length 4 with zero or one source. Thus we define $compress(G)$ for a triangle-free graph G with a 2-cell embedding on torus without non-contractible 4-cycles that will return a compressed subgraph of G that is 3-colorable exactly if G is 3-colorable:

1. Initialize H as the $(2,2)$-extension of G, let ϕ be an outer face of H and let $H^\star$ be the dual of H.
2. Find a spanning tree T of $H^\star$ and compute sz on T.
3. Let $\vec{G}$ be G oriented with the constant-degree orientation algorithm
4. For every vertex v in $V(\vec{G})$
   - Iterate through all paths P of length 4 in $\vec{G}$ starting in v. If P ends at v, do $eliminate(interior(P, G, T, sz), \vec{G})$.
   - Iterate through all pairs of paths of total length equal to 4 in $\vec{G}$ starting in v. If P_1 and P_2 end in the same vertex, reverse P_1 and do $eliminate(interior(P_1 \cup P_2, G, T, sz), \vec{G})$
   - For every edge $e := (u_i, v)$ entering v from $E(\vec{G})$.
     - For every $(u_i, a) \in E(\vec{G}), a \neq v$
       - Add u_i to $in_v(a)$.
       - For every pair $x, y \in in_v(a), x \neq y$ do $eliminate(interior(vxay, G, T, sz), \vec{G})$, updating the sets of $in_v(a)$.
   - Clear in_v.
5. Remove every edge from G such that $\{e, opp(e)\} \cap E(\vec{G}) = \emptyset$.
6. Remove all faces and vertices removed in $\vec{G}$.
7. Return G.

Finally, let us define the algorithm $threecol_5(G)$ for checking 3-colorability of a triangle-free graph G with an embedding on a torus and no non-contractible cycles of length less than five.

1. If $|V(G)| - |E(G)| + |F(G)| = 2$, the graph is planar and thus is 3-colorable, return "true".
2. Let $H = compress(G)$.
3. If H has a face of length different from 4, return "true".
4. return $threecol_4(H)$.

As we have already proven, $compress(G)$ has linear time complexity. The 3-coloring algorithm from chapter 6 has linear time complexity for quadrangulations.

From the analysis it is clear that this algorithm runs in linear asymptotic time for each graph satisfying the requirements stated in this chapter, i.e. triangle-free, embedded on torus and with no non-contractible cycles of length four.

Chapter 7

Coloring algorithm

Theoretical basis

In this chapter we will extend the previous algorithm to the case of the graphs with non-contractible 4-cycles. For this case Theorem 7.3 does not apply, however there is a theorem that does not need this assumption and is similar to 7.3, albeit more complicated:

From now on we will call a face of length l an *l-face*.

Let G be a triangle-free graph with a 2-cell embedding on a torus. We call G an *almost-quadrangulation* if there are

- no faces of length greater than 4, or
- two 5-faces, or
- four 5-faces, or
- one 6-face and two 5-faces, or
- one 7-face and one 5-face

in G and the rest of the faces in G have length 4. Note that every almost-quadrangulation is a near-quadrangulation

Theorem 7.1. *Let G be a triangle-free graph with a 2-cell embedding on a torus. If G is not 3-colorable, there is a subgraph Q of G that is an almost-quadrangulation and is not 3-colorable.*

Analogously to the previous chapter, we need to perform some operation on the graph G, so that the resulting graph H is 3-colorable if and only if G is 3-colorable, at the same time H is either an almost-quadrangulation or is 3-colorable. For this we first need to define some utilities, based on the ideas of [12]

Let G be a triangle-free graph with a 2-cell embedding on torus and let ϕ be a face of G. The face ϕ is considered *k-free* if there is no contractible cycle of length at most $\min(k, |\phi| + 1)$ containing ϕ, with the exception of the cycle bounding ϕ if ϕ is bounded by a cycle.

Lemma 7.2 ([13])**.** *Let H be a plane graph with the outer face bounded by a cycle C. If there is a non-outer face ϕ in H with length at least $|C| - 1$ that is not contained in a cycle of length at most $|C| - 2$, then every 3-coloring of C can be extended to a 3-coloring of H.*

Let G be a triangle-free graph with a 2-cell embedding on a torus, we define *freeing* of a face ϕ in G as the following procedure $free(G, \phi)$:

1. While ϕ is not 7-free:
    - Find a shortest contractible cycle C containing ϕ that is not the boundary of ϕ. Secondarily choose one with the maximal interior by inclusion.
    - Eliminate the interior of C, set the new face as ϕ.
2. return ϕ.

There will be constantly many iterations in this loop, because a shortest cycle we chose is also maximal by inclusion and after every iteration the length of the shortest non-facial cycle containing ϕ will strictly increase. To finish this algorithm we need to be able to find such cycle in asymptotically linear time.

Finding a minimal enclosing cycle

In this subsection we will show a way to find a shortest cycle containing a face ϕ in a graph G with a 2-cell embedding on a torus, secondarily choosing the maximal by inclusion of the interior. First, let us transform this problem into the same problem but for a plane graph:

Observation 7.3. *Let G be a graph with a 2-cell embedding on a torus and let ϕ be a face in G. Let H be the (3,3)-extension of G and let ϕ_H be a face in H, such that the boundary of ϕ_H is the unfolding of the boundary of ϕ in H. Let $C \subseteq G$ be a contractible cycle of length at most 7 and let C_H be the unfolding of C in H. Then ϕ is contained in the interior of C exactly if ϕ_H is contained in the interior of C_H.*

Therefore find a shortest cycle containing a face ϕ in G we can find a shortest cycle C_H containing ϕ in the (3, 3)-extension H of G.

Finding a shortest cycle containing a face ϕ in a plane graph H with the dual $H^\star$ and the outer face ϕ' can be done by finding a minimal cut between $dual(\phi')$ and $dual(\phi)$. The minimum cut is found using the minimum cut/maximum flow duality and the maximum flow is found with Ford-Fulkerson algorithm. If ϕ is not 7-free, the size of the minimum enclosing cycle, and thus the size of the minimum cut will be at most 7, therefore once we find a flow of size 8 or more, we conclude that ϕ is 7-free and stop the procedure.

The next requirement for the desired cycle is that it has to be maximal by inclusion among shortest ones, i.e. we need to maximize the amount of faces in the interior of the cycle or minimize the amount of faces outside of the cycle. For this we can choose a minimum cut that will be *the closest* to the source in the dual graph: Let a minimal cut E divide the vertices of the dual graph $H^\star$ into two parts $A(E)$ and $B(E)$. Let C be the dual of E, since E is a minimal cut between two vertices, C is a cycle. Assume that the source lies in $A(E)$. Note that the source is dual to the outer face in H, and then $A(E)$ is the set of vertices dual to those faces in H, that lie outside of the interior of C. Therefore it makes sense to minimize $A(E)$ to maximize the interior of C.

Let e be an edge from the dual of a graph in DCEL representation, define *residual capacity* of e as $cap(e) - flow(e)$.

Let $S \subseteq V(H^\star)$ be the set of all vertices in $H^\star$ that are reachable from the source by the edges with non-zero residual capacity. Since Ford-Fulkerson algorithm finished, the target does not lie in S and $S \neq V(H^\star)$. Then the minimum cut closest to the source would be a cut E' between S and $V(H^\star) \setminus S$. Indeed, in any minimum cut E all edges have zero residual capacity and no vertex reachable from the source by the edges with non-zero residual capacity will be in $B(E)$, therefore $S \subseteq A(E)$ for every minimal cut E.

Lastly, we can exclude the boundary of ϕ by choosing a specific edge from the boundary and setting its capacity to $+\infty$. We do not know which edge does not appear in the optimal cycle, so we try this with every edge and take the best result. If $|\phi| \geq 8$, we do not need to exclude the boundary as we are only interested in cycles of length at most seven.

First we will define a procedure $mincut(H^\star, v_s, v_t)$ that will find a minimum size cut in $H^\star$ between v_s and v_t, secondarily being the closest to v_s:

1. Run Ford-Fulkerson DFS relaxations from v_s to v_t for at most 7 times or until there are no augmenting paths.

2. If there are no augmenting paths from v_s to v_t:

    - Set $S \subseteq V(H^\star)$ as the set of vertices reachable from v_s by the edges with non-zero residual capacity.

- Set E as the cut between S and $V(H^\star) \setminus S$ in $H^\star$.

3. Otherwise $E := \emptyset$.

4. return E.

Also we now can define $free(G, \phi)$ in more detail:

1. Repeat:

   - Let H be the (3,3)-extension of G.
   - Let $H^\star$ be the dual of H.
   - Let C be the walk bounding ϕ.
   - Let $C_H \subseteq H$ be the unfolding of C.
   - Let ϕ_s be an outer face in H, $v_s := dual(\phi_s)$.
   - Let ϕ_t be a face in H bounded by C_H, $v_t := dual(\phi_t)$.
   - Set $C_{\min} := \emptyset$.
   - If $|C| > 7$:
     - Set $C_{\min}$ as the dual of $mincut(H^\star, v_s, v_t)$.
   - Otherwise, for every edge e from C_H:
     - Set $cap(dual(e))$ to $+\infty$.
     - Set C as the dual of $mincut(H^\star, v_s, v_t)$.
     - If C is not empty:
       - If $C_{\min}$ is empty or $|C| < |C_{\min}|$, $C_{\min} := C$.
     - Set $cap(dual(e))$ back to 1 and reset the flows in $H^\star$.
   - If $C_{\min}$ is empty, return.
   - Map $C_{\min}$ to a cycle C in G, eliminate the interior of C.
   - Set the new face in G as ϕ.

2. return ϕ.

From the analysis above we can see that $free(G, \phi)$ has time complexity linear to the size of G.

Turning into an almost-quadrangulation

In this subsection we will find a way to turn a triangle-free graph G with a 2-cell embedding on a torus into an almost-quadrangulation using freeing and a similar operation.

The next lemma will show an important property of freeing:

Lemma 7.4. *Let G be a triangle-free graph with an embedding on torus, let ϕ be a face in G, and let C be a shortest cycle containing ϕ in the interior that is not the boundary of ϕ. If ϕ is not 7-free, then $G \odot C$ is 3-colorable if and only if G is 3-colorable.*

Proof. Since $G \odot C$ is a subgraph of G, if $G \odot C$ is not 3-colorable, then G is not as well.

Otherwise note that $|C| \leq \min(7, |\phi| + 1)$, so $|C| - 1 \leq |\phi|$. At the same time, since C is a shortest cycle containing ϕ except the boundary, for every cycle C' containing ϕ in the interior but not being the boundary of ϕ, $|C'| \geq |C|$ holds and for the boundary $|\phi| \geq |C| - 1$ holds. Therefore, a coloring of C can be extended to the coloring of the interior of C according to Theorem 8.2. □

An important property of 7-free faces of length more than 5 is that they appear in every almost-quadrangulation subgraph:

Observation 7.5. *Let G be a triangle-free graph with a 2-cell embedding on torus, and let Q be a subgraph of G. Let ϕ be a 7-free face in G. If $\phi \notin F(Q)$, then the boundary of the face of Q containing ϕ is either non-contractible or has length at least $min(|\phi| + 2, 8)$.*

If $|\phi|$ is 6 or 7, the face containing ϕ in Q is either non-contractible or has length more than 7, which makes the subgraph either planar or not almost-quadrangulation. If Q is an almost-quadrangulation that is not 3-colorable, we can see that ϕ is in $F(Q)$. However, 7-free faces of length 5 in this case can still be contained inside of a face of length 7. For this case the following theorem is useful:

Theorem 7.6 ([14])**.** *Let H be a triangle-free plane graph with the outer face bounded by a cycle C of length 7. If there are two 5-faces in H not contained inside a non-facial 5-cycle, then every coloring of C can be extended to a coloring of H.*

The first step of the coloring algorithm is to find a subgraph of G, such that it is either an almost-quadrangulation or is 3-colorable and at the same time it is 3-colorable if and only if the original graph is 3-colorable. A graph G with a 2-cell embedding on a torus is *conditioned* if all faces of length 5 and more are

7-free and if no pair of 5-faces lies in the interior of a contractible 7-cycle. The act of transforming the graph into a conditioned one by freeing the faces is called *conditioning*. Now we prove that a conditioned graph is 3-colorable if it is not an almost-quadrangulation. First let us prove an auxiliary lemma:

Lemma 7.7. *Let G be a graph with a 2-cell embedding on a torus and let C be a contractible cycle in C. Then the sum of the lengths of the faces in the interior of C is even if and only if the length of C is even.*

Proof. Denote the mentioned sum as L. Every edge e in the strict interior of C (i.e. excluding the edges of C) contributes 2 to L, since the faces on both sides of e lie inside of C. Every edge from C contributes exactly 1 to L, therefore we can formulate L as $|C| + 2K$ for $K \in \mathbb{Z}$. □

Theorem 7.8. *Let G be a conditioned triangle-free graph with a 2-cell embedding on torus. If G is not an almost-quadrangulation, then G is 3-colorable.*

Proof. Let us prove it by contradiction. If G is not 3-colorable, there is an almost-quadrangulation $Q \subseteq G$ that is not 3-colorable by Theorem 8.1. We will assume that the embedding of Q derived from the embedding of G is 2-cell, as otherwise Q is planar, triangle-free and thus 3-colorable according to Grötzsch's theorem.

Let us describe the relationship between the faces of G and Q:

- Every 5-face in G will either be in Q as a face or will be inside of some 7-face in Q, where at most one 5-face in G can be inside of a 7-face in Q.

  On the other hand, let ϕ_5 be a 5-face in Q that is not a face in G, let $C_5 \subseteq Q$ be a bounding cycle of ϕ_5, then the interior of C_5 is not empty in G. However, there can not be a face of length 5 or more (since they all are 7-free) inside of C_5, therefore the interior of C_5 can only be quadrangulated, which is also impossible according to the Lemma 8.7. Consequently, every 5-face in Q will be a face in G.

- Every 6-face in G will be a face in Q according to Lemma 8.4. Every 6-face in Q is either a face in G or is quadrangulated in G according to Lemma 8.7 and arguments from the previous bullet.

- Every 7-face in G will be a face in Q (by Observation 8.5). A 7-face in Q will either be a face in G or will contain exactly one 5-face and the rest 4-faces in the interior due to Lemma 8.7.

Let S_H for a graph H with a 2-cell embedding on some surface be the multiset, defined as

$$S_H = \{|\phi| \mid \phi \in F(H), |\phi| > 4\}$$

Since S_Q is an almost-quadrangulation, S_Q can be $\emptyset$, $\{5, 5\}$, $\{5, 5, 5, 5\}$, $\{6, 5, 5\}$ or $\{7, 5\}$.

- If $S_Q = \{7, 5\}$, then S_G can be either $\{7, 5\}$ or $\{5, 5\}$.
- If $S_Q = \{6, 5, 5\}$, then S_G can be either $\{6, 5, 5\}$ or $\{5, 5\}$.
- In all other cases $S_Q = S_G$.

As we can see, all cases imply that G is an almost-quadrangulation, which cannot be true. □

The strategy is to condition the graph, and if we got an almost-quadrangulation, apply the 3-colorability algorithm for near-quadrangulations. Here we define the procedure *condition*(G) that will return a subgraph H of G that is 3-colorable if and only if G is 3-colorable, where if H is not an almost-quadrangulation, then G is 3-colorable:

1. $H := compress(G), S := \emptyset$.
2. $\mathcal{S} := \{\{5\}, \{5, 5\}, \{5, 5, 5\}, \{5, 5, 5, 5\}, \{6\}, \{6, 5\}, \{6, 5, 5\}, \{7\}, \{7, 5\}\}$ is a set of all states of S from which an almost-quadrangulation is reachable (will be clarified later).
3. While $F(H) \setminus S$ contains a face with length > 4 and while the multiset of lengths of faces in H belongs to $\mathcal{S}$:
   - Let ϕ be a face from $F(H) \setminus S$ such that $|\phi| \geq 5$.
   - $\phi = free(H, \phi)$, remove the faces deleted during this procedure from S.
   - If $|\phi| = 5$, for every 5-face ϕ' from S find a shortest cycle C in H containing ϕ and ϕ'.
     - If there is ϕ', for which $|C| = 7$:
       - Clean the interior of C, let the new face be ϕ_{new}.
       - Set $\phi_{new} := free(H, \phi_{new})$, remove the faces deleted during $free$ from S.
       - Add ϕ_{new} to S.
   - Otherwise add ϕ to S.
4. Otherwise return H.

At the first step we perform *compress*(G) even though G may contain non-contractible 4-cycles. To do this correctly, we will eliminate the interiors of only contractible 4-cycles in G, ignoring the non-contractible ones. Another difference is that Observation 7.5 does not help in case of non-contractible 4-cycles, however

it can be shown that $K_{2,9}$ will have a non-facial 4-cycle in any embedding on a torus, which means that in *compress* algorithm $|in_v(a)| \leq 9$ will hold. Compression ensures that the result of $free(H, \phi)$ will be a cycle of length at least 5, since there are no non-facial 4-cycles in H.

At every step of the algorithm it is true that all faces in S are 7-free and no pair of 5-faces from S lie inside of a contractible 7-cycle, at the end of the algorithm all faces of length more than 4 will be in S. After each iteration at step 3 a new face is added, possibly removing some other faces lying inside of the new one from S. However, since the aforementioned invariants hold, the only time a face can be removed from S is when we find a pair of 5-faces contained in a 7-cycle or when the result of $free$ is a 7-face, so during cleaning the interior a 5-face might be deleted. Furthermore, only two 5-faces can be deleted during the first case and at most one - during the second case and in both cases a 7-face is added to S.

From this we can see that states of S can not "loop": after each iteration either the size of S increases or a new 7-face is added, and once there are two 7-faces in S, H will definitely not become an almost-quadrangulation. More generally, this fact allows us to define a set of states of S, from which it is possible to obtain an almost-quadrangulation, this set being $\mathcal{S}$. These states with all possible transitions are depicted in Figure 16, where solid line transitions stand for freeing a found face and dashed ones stand for finding a pair of 5-faces inside of a 7-cycle.

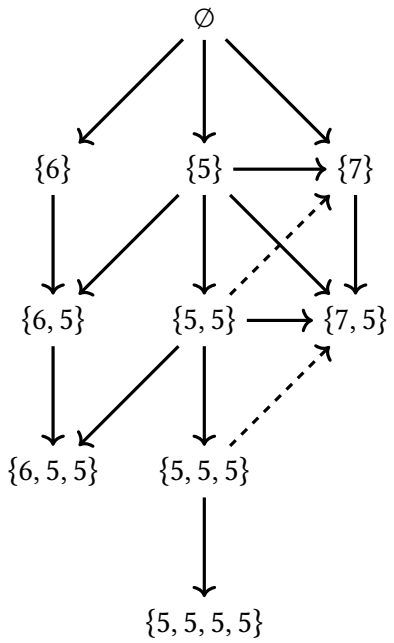


Figure 7.1 A graph of all valid states of S and transitions between them

Note that once S is not in this graph we can stop the algorithm and claim

that the graph is 3-colorable since no conditioned subgraph can be an almost-quadrangulation.

We already know that $compress(G)$ (with slight modifications for non-contractible 4-cycles) has linear time complexity for graphs embedded on a torus, that $free(H, \phi)$ has linear time complexity and that the number of invocations of $free$ is constant. We can use the same min-cut finding algorithm as in $free$ to find a shortest cycle containing 5-faces ϕ and ϕ', except we connect $dual(\phi)$ and $dual(\phi')$ to a new auxiliary vertex that will be a target (the same multi-target flow trick we used in Chapter 3). Since we are interested only in 7-cycles, we can stop Ford-Fulkerson algorithm once the flow is 8 or more. Note that the dual of the resulted minimum cut will be a set of cycle $\mathscr{C}$ but also will have the size at most 7. Since the shortest cycle in $\mathscr{C}$ has length at least 4, it follows that $|\mathscr{C}| = 1$ and the result is an enclosing cycle. In summary, this part and the whole algorithm will take linear asymptotic time.

With this we can define the final algorithm $threecol(G)$ that determines whether a triangle-free graph G, embedded on a torus is 3-colorable:

1. If $|V(G)| - |E(G)| + |F(G)| = 2$, the embedding is not 2-cell, return "true".
2. $H = condition(G)$.
3. If H is not an almost-quadrangulation, return "true".
4. return the result of $threecol_4(H)$.

As was already shown, $condition(G)$ has time complexity linear to the size of G. Since we run $threecol_4$ on almost-quadrangulation and an almost-quadrangulation is a near-quadrangulation, the last steps also has linear time complexity. Therefore, $threecol(G)$ has linear time complexity for every triangle-free graph embedded on torus.

Chapter 8

Results

The main goal of this work was to implement the described algorithm and verify if the algorithm is actually practical, which means it runs in realistic time for big graphs and the execution time is linearly proportional to the size of this graph.

The algorithm was implemented in C++ language primarily as a library using C++17 standard. We also implemented a testing framework `threecol_test` that was used for intermediate unit-tests and the benchmarks.<

For the benchmarks we have used the results of [6], where it is stated that every triangle-free graph with an embedding on a torus that is not 3-colorable can be generated from one of 186 graphs called *templates* by quadrangulating a certain subset of faces in it (we call these faces *non-floating*). Furthermore, every such generated graph is not 3-colorable. This is convenient for testing our program, as for non 3-colorable graphs, all parts of the algorithm (such as trying all valid boundaries during the nowhere-zero flow search, trying all cycle precolorings during double cycle extension, and so on) are executed. Therefore, we create an input graph for the benchmark by taking one of the templates and quadrangulating all its non-floating faces. The problem is that some quadrangulations might create triangles in the graph. For this a certain quadrangulation technique is used:

Let ϕ be a face that needs to be quadrangulated and let C be the bounding cycle of ϕ. We can assume that $|C|$ is even, as it is impossible to quadrangulate faces of odd length according to Lemma 8.7. We add n face layers inside ϕ by creating cycles $C_1, C_2, \dots C_n$ of the same length and connecting the vertices of C_1 with C and C_{i-1} with C_i for $i \in \{2, \dots n\}$ as shown on Figure 8.1. The interior of C_n will be a new face ϕ'. Overall it is enough to create three face layers to be able to quadrangulate ϕ' arbitrarily without the risk of creating triangles or multi-edges. During the graph generation the amount of face layers inside each non-floating face is the primary method of controlling the size of the resulting graph (it might resemble spinning a web of the needed size). We can quadrangulate ϕ' trivially by creating a new vertex inside of it and connecting it to every second vertex in

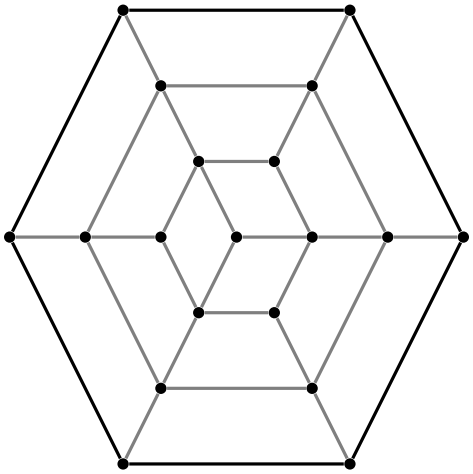


Figure 8.1 Quadrangulating a face after adding 2 face layers

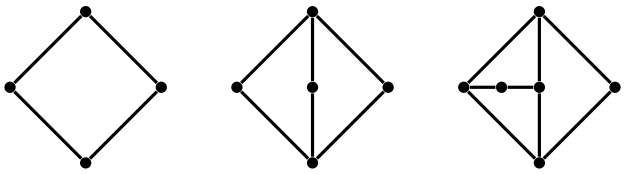


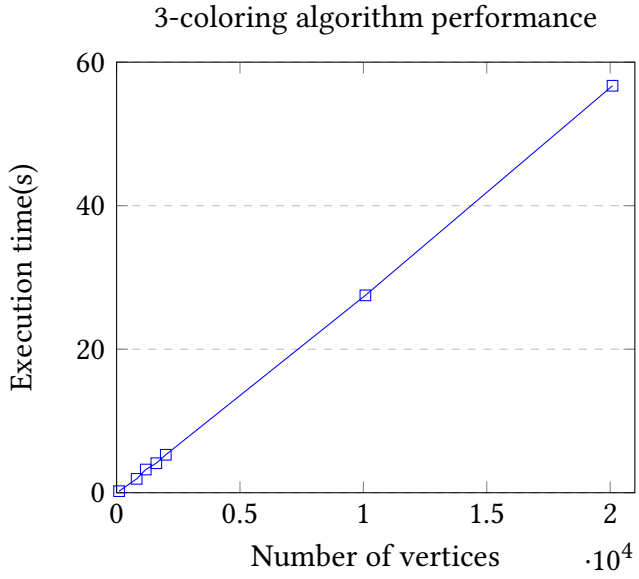
Figure 8.2 Dividing a 4-face in a random manner

the clockwise order.

This quadrangulation method is straightforward, however it has some disadvantages. An important one is that the resulting graphs are not random and a big part of the graph follows the same pattern (i.e. the repetitive "web" takes almost the whole size of the graph). This makes benchmarks dependent on the mentioned quadrangulation pattern. One of the solutions is to quadrangulate the face the following way: after quadrangulating ϕ with this method using a small number of face layers, iteratively pick random 4-faces inside of ϕ and divide the chosen faces into two 4-faces until the size of the quadrangulation is sufficient. This would diversify created graphs, but unfortunately this testing method was out of our time scope.

The benchmark was performed the following way: for a chosen number of vertices N we generated a graph G_j with roughly N vertices using the j-th template out of 186, ran the program on every G_j and averaged the result over all j. With this we obtained our estimate of the expected execution time of the implementation on a graph with approximately N vertices. The set of chosen N is $\{100, 800, 1200, 1600, 2000, 10000, 20000\}$ and the corresponding average execution time for each N can be seen in Figure 8.3:

As one can see, the expected execution time indeed grows linearly to the



size of the graph. Moreover, it takes less than a minute to process a graph with 20000 vertices. The program correctly decides that all received graphs are not 3-colorable. The program was tested on a laptop CPU with 4.5GHz tact frequency. The compiler was GNU C++ with Release-type CMake build, however the code should be compilable by any compiler supporting C++17 standard and CMake.

Another type of testing could be checking the correctness of the output on large random graphs but a proper testing would involve generating random triangle-free graphs embeddable on a torus and using another 3-coloring algorithm that would verify if the decision of our program was correct. This was out of the scope of this work, so the program was tested only with the aforementioned template extensions, several manually constructed 3-colorable graphs and occasional intermediate unit tests. On all of them the program returns correct answer.

Conclusion

In summary, the goal of implementing and evaluating the algorithm, determining if a triangle-free graph embedded on a torus is 3-colorable, was met. The algorithm was implemented fully and in numerous tests was shown to return the correct result and run in time that is linearly proportional to the size of the input graph. An explanation of the algorithm and its implementation was given in this paper and in the commentaries in the code.

The program is reasonably fast and, we believe, can be used in practice in future: it can be used in future works considering 3-coloring on graphs embedded on a surface or possibly other combinatorial problems, as this algorithm solves an NP-complete problem, even if on a limited number of inputs.

For the future work however, it would make sense to test and evaluate this implementation more. It is possible that there are still mistakes which we did not catch due to the nature of the algorithm. Since the output of a program is binary, it is difficult to judge if the answer was correct ”by luck” or some mistake was overshadowed by another mistake (which has certainly happened several times during the testing and fixing). Moreover we have found the testing inconclusive also because the program was not tested on large 3-colorable graphs. It is generally difficult to independently check if some big graph is 3-colorable and the implemented algorithm returns no coloring, except for some of its parts (for example, for near-quadrangulations), making it impossible to verify the answer.

As a future work, we leave finding a practical algorithm that would not only determine if a triangle-free graph embedded on a torus is 3-colorable but also find a proper 3-coloring.

Bibliography

[1] H. Grötzsch. "Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel". In: *Math.-Natur. Reihe* 8 (1959), pp. 109–120.

[2] C. Thomassen. "3-list-coloring planar graphs of girth 5". In: *J. Combin. Theory, Ser. B* 64 (1995), pp. 101–107.

[3] A. V. Kostochka and M. Yancey. "Ore's conjecture on color-critical graphs is almost true". In: *J. Comb. Theory, Ser. B* 109 (2014), pp. 73–101.

[4] Z. Dvořák, K. Kawarabayashi, and R. Thomas. "Three-coloring triangle-free planar graphs in linear time". In: *Trans. on Algorithms* 7 (2011), article no. 41.

[5] Z. Dvořák, D. Kráľ, and R. Thomas. "Three-coloring triangle-free graphs on surfaces I. Extending a coloring to a disk with one triangle". In: *Journal of Combinatorial Theory, Series B* 120 (2016), pp. 1–17. ISSN: 0095-8956.

[6] Z. Dvořák and J. Pekárek. "Characterization of 4-critical triangle-free toroidal graphs". In: *Journal of Combinatorial Theory, Series B* 154 (2022), pp. 336–369. ISSN: 0095-8956.

[7] Z. Dvořák and J. Pekárek. "Coloring near-quadrangulations of the cylinder and the torus". In: *European Journal of Combinatorics* 93 (2021), p. 103258. ISSN: 0195-6698.

[8] W. Tutte. "A Contribution to the Theory of Chromatic Polynomials". In: *Canadian Journal of Mathematics* 6 (1954), pp. 80–91.

[9] D. Eppstein. "Dynamic generators of topologically embedded graphs". In: *arXiv preprint cs/0207082* (2002).

[10] Ł. Kowalik and M. Kurowski. "Oracles for bounded length shortest paths in planar graphs". In: *ACM Trans. Algorithms* 2 (2006), pp. 335–363.

[11] V. A. Aksionov. "On continuation of 3-colouring of planar graphs". In: *Diskret. Anal. Novosibirsk* 26 (1974). In Russian, pp. 3–19.

[12] Z. Dvořák, D. Král', and R. Thomas. "Three-coloring triangle-free graphs on surfaces VII. A linear-time algorithm". In: *Journal of Combinatorial Theory, Series B* 152 (2022), pp. 483–504.

[13] Z. Dvořák, D. Král', and R. Thomas. "Three-coloring triangle-free graphs on surfaces IV. Bounding face sizes of 4-critical graphs". In: *Journal of Combinatorial Theory, Series B* 150 (2021), pp. 270–304.

[14] V. A. Aksenov, O. V. Borodin, and A. N. Glebov. "Extending 3-colorability from a 7-face to a planar triangle-free graph". In: *Sib. Elektron. Mat. Izv.* 1 (2004). In Russian, pp. 117–128.

Appendix A

Using the algorithm

The program is implemented in C++ and was tested mainly in native Linux environment and in Windows Visual Studio IDE. It can be used as a library, where the source files are at `threecol/include/` or as an executable.

In case one wants to use the library, the main framework is described in `test.hpp`. The input graph is an adjacency list `g` that is transformed into DCEL with `build_from_list`. Each vertex in the input graph is identified with a number starting from 0, `g[i]` is a vector of all edges incident to `i` sorted clockwise. An edge (u, v) appears in `g[u]` as `simple_edge(u, v, index_1, shared)` and in `g[v]` as `simple_edge(v, u, index_2, shared)` where `index_1` and `index_2` should be unique for all instances of `simple_edge` in `g` and `shared` should be unique for every such pair representing an edge.

To use the executable, one needs to build the project: on Linux you can use `test_all.sh` from the root directory of the project or can build it on their own with `cmake .` and `make`. In `threecol/threecol-test/` of the build directory there will be two executables: `threecol-big-test` and `threecol-manual`.

`threecol-big-test n` reads `output_n.txt`, quadrangulates the non-floating faces and runs the 3-coloring algorithm while measuring the execution time. `threecol-big-test` without arguments does the same for all `output_*.txt` templates and returns the mean execution time and number of vertices. If you want to change the size of the quadrangulation, you can set a custom number of face layer in `QUAD_FACTOR`.

`threecol-manual` expects on `stdin` the following description of the graph:

1. On the first line a number n - the number of vertices in the graph. All vertices have indices from 0 to $n - 1$.

2. On the line $i + 2$ there is sz_i - the number of vertices incident to the vertex i followed by sz_i numbers - indices of the vertices incident to i ordered clockwise around i.

An adjacency list of K_4.

```
std::vector<std::vector<simple_edge>> g(4);
g[0] = {simple_edge(0, 1, 0, 0),
    simple_edge(0, 2, 1, 1),
    simple_edge(0, 3, 2, 2)};
g[1] = {simple_edge(1, 2, 3, 3),
    simple_edge(1, 0, 4, 0),
    simple_edge(1, 3. 5, 5)};
g[2] = {simple_edge(2, 3, 6, 4),
    simple_edge(2, 0, 7, 1),
    simple_edge(2, 1, 8, 3)};
g[3] = {simple_edge(3, 1, 9, 5),
    simple_edge(3, 0, 10, 2),
    simple_edge(3, 2, 11, 4)};
```

3. For the case of `output_*.txt` there are more lines. They describe the faces that need to be quadrangulated. One line is a description of one face, a sequence of vertex indices that are on the boundary ordered counterclockwise. This is not the part of the input in `threecol-manual`.

All graphs in `output_*.txt` fit this description.

One more executable `showgraph` is used to read the templates in `template.txt` and convert them to the structure readable by our program in `output_X.txt` for X from 1 to 186. These new files are used as the templates in performance testing. This program as well as the list of templates are taken from [6]. To build this executable and generate the test data locally, build the project with `cmake . -DROLL_TESTS=True`, otherwise the existing in `threecol/threecol-test/tests` templates are generated. The option `-DROLL_TESTS=True` has been shown to work reliably only on Linux.