Graph Embedding Algorithms

by

ANDREI GAGARIN

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada

© Copyright by Andrei Gagarin, 2003

June 11, 2003

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

GRAPH EMBEDDING ALGORITHMS

BY

ANDREI GAGARIN

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

ANDREI GAGARIN © 2003

Permission has been granted to the Library of The University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to University Microfilm Inc. to publish an abstract of this thesis/practicum.

The author reserves other publication rights, and neither this thesis/practicum nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

Abstract

A *topological surface* S can be obtained from the sphere by adding a number of handles and/or cross-caps. Any topological surface can be represented as a polygon whose sides are identified in pairs. The *projective plane* can be represented as a circular disk with opposite pairs of points on its boundary identified. The *torus* can be represented as a rectangle with opposite sides of its boundary identified.

Given a graph G and a topological surface S, we ask whether it is possible to draw the graph on the surface without edge crossings. Such a drawing of G on the surface is called an *embedding* of G in S. It divides the surface into connected regions called *faces*. An embedding is 2-*cell* if each face is equivalent to an open disk.

Efficient embedding algorithms for the plane are well-known. By Kuratowski's Theorem, a non-planar graph G contains a subdivision of K_5 or $K_{3,3}$ as a subgraph. The objective of this thesis is to devise efficient practical embedding algorithms for the projective plane and torus.

The major contributions of the thesis are:

- A new linear time algorithm to detect a projective planar graph;

- Given a K_5-subdivision in G, a linear time algorithm to determine if G is toroidal or to provide a $K_{3,3}$-subdivision in G;

- Simple methods to transform a planar embedding into a 2-cell projective planar or toroidal embedding.

The known linear time algorithm for the projective plane in [28] appears to be infeasible and it is not clear if the approach is correct. The practical linear time projective planarity algorithm of the thesis improves the $O(n^2)$ time algorithm of [30]. The algorithm for the torus permits to reduce toroidality testing to a constant number of planarity checks or to a $K_{3,3}$-subdivision in the graph. It runs in linear time and can be used to simplify algorithms presented in [21] and [31].

Acknowledgements

I am very thankful to my advisor, Professor William Kocay, for involving me in algorithmic and topological graph theory research. Without his experience, intuition, calm, confidence and understanding this thesis definitely would not be possible. I hope we will collaborate on these nice problems in the future.

I would like to thank:

- my parents and friends overseas for their direct or indirect moral support and encouragement;
- my former scientific advisors, Professors Igor E. Zverovich, Regina I. Tyshkevich, Pierre Duchet, Charles Payan and Lynn Batten for showing me the taste of real research and for introducing me to such a wonderful research area as Graph Theory and Combinatorics;
- The Department of Mathematics at the University of Manitoba for providing me the opportunity to study in Winnipeg;
- The Fields Institute (University of Toronto) and the Department of Combinatorics and Optimization (University of Waterloo) for inviting me to the Special Year 1999-2000 on Graph Theory and Combinatorial Optimization at The Fields Institute;
- Professor Helen Cameron and Administrative Assistant Lynne Romuld

(Department of Computer Science) for their understanding and help in difficult situations during my studies at the University of Manitoba.

List of Figures

Contents

Chapter 1

Introduction

One of the fundamental classical problems in modern graph theory and combinatorics is the problem of embedding graphs in topological surfaces. A *topological surface* can be obtained from the sphere by adding a number of handles or crosscaps. A *graph* is a pair $G = (V, E)$ such that V is a set and E is a subset of $V^{(2)}$. The elements of V are the *vertices* or *points* of G and the elements of E are its *edges* or *lines*. The notation $\{u, v\}$ or uv is used to denote an edge of G. Usually a graph is pictured by drawing a dot for each vertex and a line joining two corresponding dots for each edge. Given a graph, one wants to draw it on a surface without edge crossings whenever it is possible.

One of the initial results in graph theory is a structural characterization of pla-

nar graphs through excluded subdivisions of K_5 and $K_{3,3}$ by K. Kuratowski [25]. This was published in the 1930s. However efficient algorithms to recognize if a graph is planar appeared much later. For example, the popular linear time planarity-testing algorithm by J. Hopcroft and R. Tarjan [19] was published only in 1974. The *orientable* (*non-orientable*) *genus* of a graph is the smallest orientable (non-orientable) genus of a surface in which the graph can be embedded. In general, the problem of finding the genus of a graph was proved to be NP-complete by C. Thomassen [36].

Recently, Kuratowski's characterization of planar graphs was generalized for non-orientable surfaces by D. Archdeacon and J.P. Huneke [2] and for orientable surfaces by R. Bodendiek and K. Wagner [3]. In a series of papers on graph minors, N. Robertson and P. Seymour [35] generalized Kuratowski's result for an arbitrary surface. This implies that for a given surface the question of whether a graph is embeddable into the surface can be answered in polynomial time.

Moreover, B. Mohar [29] claimed to develop a series of linear time algorithms to answer the question. Unfortunately, these linear time algorithms appear to be infeasible, and are more of theoretical interest than practical. The descriptions of [28], [21] and [29] are missing many of the details necessary for an implementation of them. It is not clear if the approach is correct and covers all the cases providing a linear time algorithm. However, as it is mentioned in [41], the description of [28] gives some insights into the problem. The only known

efficient implemented algorithm is the $O(n^2)$ projective planarity-checking algorithm by W. Myrvold and J. Roth [30].

This thesis is focussed on devising linear time practical algorithms to determine if there exists an embedding of a graph in the projective plane and/or torus. These are the topological surfaces closest to the plane. Early algorithms for these surfaces described in [12] and [32] are known to be wrong (personal communication by W. Myrvold). Many known algorithms for the projective plane and torus (eg. [30], [28] and [21]) begin with a Kuratowski subgraph K_5 or $K_{3,3}$ in a graph, and try to extend an embedding of K_5 or $K_{3,3}$ to an embedding of the whole graph on the corresponding surface. For a graph G containing a K_5-subdivision, this thesis presents new algorithms to reduce the projective planarity or toroidality testing of G to a constant number of planarity checks or to a $K_{3,3}$-subdivision in G. For a graph G containing a $K_{3,3}$-subdivision, the thesis provides a new detailed algorithm to tell if G is projective planar. In summary, we have devised a new linear time algorithm to detect a projective planar graph and a linear time algorithm that either determines the toroidality of a graph or returns a $K_{3,3}$-subdivision in it.

Chapter 2 provides basic notation, definitions and results related to the problem and algorithms. Chapter 3 describes the main ideas of the Hopcroft-Tarjan planarity algorithm. The ideas and concepts of the planarity algorithm are used in different forms for other algorithms in the thesis. Necessary conditions for a 2-cell embedding of a graph on the projective plane and torus are given

in Chapter 4. Section 4.3 presents methods for transforming a planar embedding into a 2-cell embedding on the projective plane and torus. The methods are another main contribution of the thesis. They are used in the software *Groups&Graphs* [24].

In Chapter 5, we describe known implemented general algorithms for the projective plane and torus from [30] and [31]. These algorithms help us to better understand surfaces with respect to the problem and its practical solution. The projective planarity checking algorithm of W. Myrvold and J. Roth has $O(n^2)$ time complexity, whereas the toroidality checking algorithm is exponential in the worst case.

We have completely characterized projective planar and toroidal embeddings of certain kinds of graphs containing a K_5-subdivision and developed linear time algorithms to tell if the graphs are projective planar or toroidal. Structural results for graphs containing a subdivision of K_5 are presented in Chapter 6. Given a non-planar graph G with a subdivision of K_5 as a subgraph, we can either transform the K_5-subdivision into a $K_{3,3}$-subdivision in G, or else we obtain a partition of the vertices of $G\backslash K_5$ into equivalence classes. As a result, we can reduce a projective planarity or toroidality algorithm to a small constant number of planarity checks as in [19], or to a graph G containing a $K_{3,3}$-subdivision. The corresponding new algorithms are described in Section 7.1 for the projective plane and in Chapter 9 for the torus. Our new algorithms are reasonable to implement. This approach significantly simplifies algorithms

presented in [21], [28] and [30]. We then need to consider only the embeddings on the given surface of a $K_{3,3}$-subdivision, which are much less numerous and more symmetric than those of K_5. Also our new linear time algorithm of Chapter 9 can be used to restrict the exponential algorithm of [31] to graphs containing a $K_{3,3}$-subdivision.

A description of our new linear time projective planarity algorithm is presented in Chapters 7 and 8. This algorithm is more efficient than the $O(n^2)$ time algorithm of [30] described in Section 5.1. Chapter 7 describes structural results and all possible cases to complete a $K_{3,3}$-subdivision to an embedding of a graph G in the projective plane. We consider a spanning subdivision of $K_{3,3}$ in the graph. A case of a non-spanning $K_{3,3}$-subdivision in G can be treated recursively by using the recursion ideas of the Hopcroft-Tarjan planarity algorithm.

Recent results of [11] suggest an efficient method to compute the orientable genus for a graph embedded in the projective plane. Our projective planarity algorithm can be used as a preliminary step to use the approach of [11].

A graph embedding can be used to design a VLSI layout. Given a VLSI to design, we can represent its elements and wire connections by vertices and edges of a graph. Since connections between elements should not cross, we are interested in a drawing of the graph without edge crossing. This provides a practical motivation to obtain a graph embedding with particular properties.

Chapter 2

Graphs and Surfaces: Basic Notation and Results

Basic graph-theoretic terminology in this thesis follows Bondy and Murty [4] and Diestel [8]. A graph $G = (V, E)$ is *undirected* if the edges of G are unordered pairs of vertices and G is *simple* if there are no multiple edges or loops. A graph $G = (V, E)$ is 2-*connected* if for any two vertices, $u, v \in V$, there are two internally disjoint paths in G with endpoints u and v. In other words, any two vertices $u, v \in V$ are on a cycle in G.

In this thesis we consider the graph embedding problem for 2-connected, undirected, simple graphs. For graphs that are not 2-connected we can decide on

their embedding in the plane, projective plane or torus by considering their maximal 2-connected subgraphs.

Chapter 2 describes the polygon representation of the surfaces, defines an embedding of a graph and related things. Finally, the chapter describes basic results related to the graph embedding algorithms.

2.1 Basic Notation and Definitions

The description of topological closed surfaces is taken from [13]. The only topologically distinct (i.e. non-homeomorphic) types of *closed orientable surfaces* are the sphere, the torus, and, in general, the generalized torus with p holes or the sphere with p handles ($p = 1, 2, 3, \ldots$). For *closed non-orientable surfaces*, the only topologically distinct types are given by the sphere with q cross-caps ($q = 1, 2, 3, \ldots$).

According to [13], any closed surface S can be constructed from a curvilinear polygon homeomorphic to a circular disk by identifying sides in pairs. Each side from a pair is denoted by the same indexed symbol and is oriented on the polygon boundary. We use the superscript " $+$ " to denote a clockwise orientation of a side, like a^+, and the superscript " $-$ " to denote a counter-clockwise orientation of a side, like a^-, on the polygon boundary. Also it can be proved that any surface S can be decomposed into such a polygon.

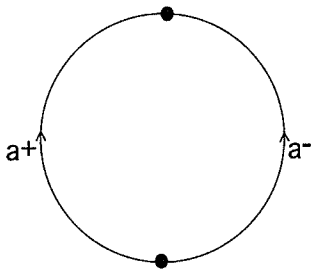

Figure 2.1: The polygonal representation of the sphere

Theorem 2.1 ([13]) *Any surface S can be obtained from a polygon homeomorphic to a circular disk by identifying pairs of sides denoted by the same symbol where the symbolic side representation of the cyclic polygon boundary is one of the following types:*

(i) a^+a^- *(the sphere),*

(ii) $a_1^+b_1^+a_1^-b_1^-a_2^+b_2^+a_2^-b_2^- \ldots a_p^+b_p^+a_p^-b_p^-$, *(the sphere with p handles),*

(iii) $a_1^+a_1^+a_2^+a_2^+ \ldots a_q^+a_q^+$, *(the sphere with q cross-caps).*

The polygon representation of the sphere is depicted in Figure 2.1. We obtain the sphere from the circular disk by identifying its two sides a^+ and a^-.

Since the sphere is equivalent to the plane, the polygon representation of the sphere is not used to embed graphs on the surface. However it is convenient to consider other topological surfaces as polygons. This simplifies the surface

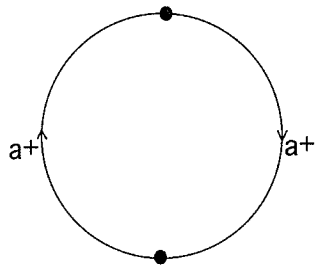

Figure 2.2: The polygonal representation of the projective plane

representation and drawings on the surface. Also an embedding in the interior of the polygon is planar and we need only consider combinatorially its boundary to decide on an embedding.

The polygon representation of the projective plane is depicted in Figure 2.2. We obtain the projective plane from the circular disk by identifying its two sides a^+ and a^+. This is equivalent to identifying opposite points on the circular disk boundary. Therefore we will consider the projective plane as a circular disk with opposite pairs of points on its boundary identified.

The polygon representation of the torus is depicted in Figure 2.3. We can obtain a cylinder from the rectangle by identifying two opposite sides a^+ and a^-, or b^+ and b^-. Then we obtain the torus by identifying the remaining opposite sides. We will consider the torus as a rectangle with the opposite sides having opposite orientation on its boundary identified.

The cylinder is an intermediate surface for the torus construction and it can

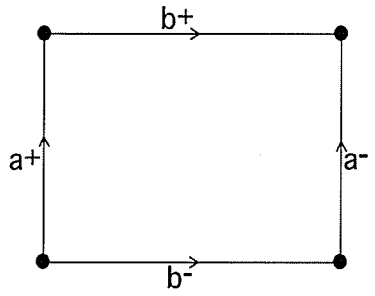

Figure 2.3: The polygonal representation of the torus

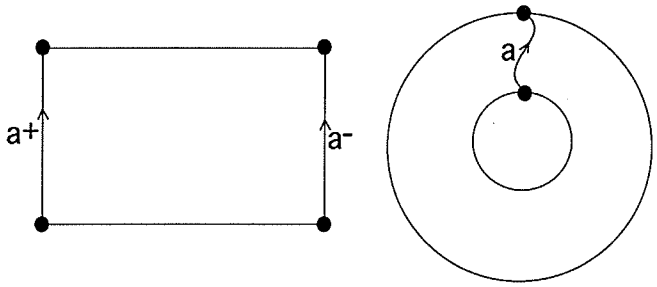

Figure 2.4: The plane drawing of the cylinder

be depicted in the plane as a cyclic band (see Figure 2.4) or as a rectangle with one pair of opposite sides identified. The cylinder and its planar properties play an important role in the torus embedding algorithms.

Given a graph G, we want to determine if G can be drawn on a surface S without edges crossing. This problem is known as the *graph embedding problem.*

Definition 2.1 An *embedding* $\psi_S(G)$ of graph G on a surface S is a mapping of

its vertices into distinct points of the surface S, and its edges into simple curves on S meeting only in common endpoints. A graph G is called *embeddable* on the surface S if it admits an embedding on S. Otherwise G is *non-embeddable* on S.

Definition 2.2 For an embedding $\psi_S(G)$ of graph G on a surface S, the connected regions of $S \backslash G$ are called the *faces* of the embedding.

A *walk* in a graph G is a non-empty alternating sequence $v_0 e_0 v_1 e_1 \ldots e_{k-1} v_k$ of vertices and edges in G such that $e_i = \{v_i, v_{i+1}\}$ for $0 \leq i < k$. If $v_0 = v_k$, the walk is *closed.* Every face is an open set on the surface bounded by edges and vertices of the graph. The *closure* of a face contains the edges and vertices of the graph on its boundary. To every face then corresponds its boundary, which is a closed walk in the graph.

A graph G embeddable on the sphere is planar since the sphere is equivalent to the plane plus one point at infinity (for example, see [13]). Clearly, a planar embedding of the graph G can be drawn on any other surface as well. However a planar embedding of G on the torus would have a face that is not equivalent to an open disk and it does not completely "fit" the surface. Therefore we are interested in embeddings of G that use the surface in full.

Definition 2.3 An embedding of G on the surface S is a 2-*cell embedding* if each face of the embedding is homeomorphic to an open disk.

The algorithms described in the thesis are for constructing a 2-cell embedding of the graph G. Given a 2-cell embedding of G on a surface S, the following definition provides a description of the faces of the embedding and their interrelations with the graph edges.

Definition 2.4 Given a 2-cell embedding $\psi_S(G)$ of G on the surface S, the *dual graph* $\psi_S^*(G) = (V^*, E^*)$ of the embedding $\psi_S(G)$ is defined as:

(i) each face f of $\psi_S(G)$ is a vertex $f \in V^*$ of $\psi_S^*(G)$;

(ii) for each edge $e \in G$ on the boundary of faces f_1 and f_2 of $\psi_S(G)$ (it might be that $f_1 = f_2$), there is an edge $e^* = \{f_1, f_2\} \in E^*$ corresponding to e.

By the definition, a dual graph $\psi_S^*(G) = (V^*, E^*)$ can be considered as embedded on the same surface S. Then the dual graph of the embedding of $\psi_S^*(G)$ naturally gives back $\psi_S(G)$.

Definition 2.5 A *rotation system* of a graph is a set of cyclically ordered adjacency lists of its vertices.

A rotation system provides a combinatorial description of a graph embedded on a surface and can be constructed by an algorithm. For a non-orientable surface, the rotation system also includes a signature for every edge. The signature can be defined as follows.

Definition 2.6 The *signature* of an edge in a rotation system is $+1$ or -1. It is negative when the edge "crosses the boundary" of a non-orientable surface and positive otherwise.

A more formal description of a rotation system and the signature is provided in [17].

Definition 2.7 Two embeddings $\psi_S^1(G)$ and $\psi_S^2(G)$ of graph G are *combinatorially equivalent* if there is an isomorphism from $\psi_S^1(G)$ to $\psi_S^2(G)$ respecting or reversing the rotation system.

An embedding of a graph on a surface is described by a rotation system only up to a continuous transformation of the surface that does not cut the surface. For example, the transformation of the torus known as a *Dehn twist* (see [18]) can give two combinatorially equivalent, but distinct non-isotopic embeddings as in [15]. A cycle on the torus is *essential* if it cannot be contracted continuously on the torus into a point. The Dehn twist consists of cutting the torus along an essential cycle to create a cylinder. Then one end of the cylinder is given one full twist, and the ends are glued back together to create a torus. The Dehn twist demonstrates that a rotation system does not always specify an embedding completely.

Consider the embedding of $K_{3,3}$ on the projective plane of Figure 2.5 as an example of a graph embedding. The rotation system for the embedding of

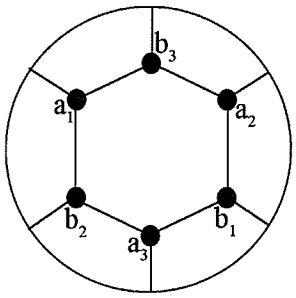

Figure 2.5: An embedding of $K_{3,3}$ on the projective plane

Figure 2.5 is:

$a_1 \rightarrow (b_1, -1), (b_3, +1), (b_2, +1),$

$a_2 \rightarrow (b_2, -1), (b_1, +1), (b_3, +1),$

$a_3 \rightarrow (b_3, -1), (b_2, +1), (b_1, +1),$

$b_1 \rightarrow (a_1, -1), (a_3, +1), (a_2, +1),$

$b_2 \rightarrow (a_2, -1), (a_1, +1), (a_3, +1),$

$b_3 \rightarrow (a_3, -1), (a_2, +1), (a_1, +1).$

Definition 2.8 A *subdivision* of a graph G is a graph that can be obtained from G by substituting paths of non-zero length for its edges. We denote a subdivision of G by TG.

Informally, a subdivision TG can be viewed as produced by successively adding some new vertices of degree 2 on the edges of $G = (V, E)$. In graph-theoretical terms adding a new vertex of degree 2 on an edge $\{u, v\} \in E$ corresponds to

deleting the edge $\{u, v\}$ from E, and then adding a new vertex w into V and two new edges $\{u, w\}$ and $\{w, v\}$ into $E\backslash\{u, v\}$. Clearly, if G is embeddable on a surface S, then any subdivision TG is also embeddable in S and vice versa.

2.2 Overview of the Related Results

Any topological surface can be considered as obtained from the sphere by adding a number of handles and/or crosscaps (see [37]). When we add $h \geq 0$ handles to the sphere S_0 we obtain the orientable surface S_h. When we add $k \geq 1$ crosscaps to S_0 we obtain the non-orientable surface N_k. Thus S_1 denotes the torus, N_1 denotes the projective plane and N_2 denotes the Klein bottle. The following theorem relates a surface S to a 2-cell embedding of a graph G on S.

Theorem 2.2 (Euler's formula, [37]) *Let G be a connected graph with n vertices and m edges having a 2-cell embedding with f faces on a surface S. Then S is homeomorphic either to the orientable surface S_h, where h is defined by the equation*

$$n - m + f = 2 - 2h;$$

or S is homeomorphic to the non-orientable surface N_k, where k is defined by the equation

$$n - m + f = 2 - k.$$

The equation of Theorem 2.2 is usually known as *Euler's formula* for a graph G which is 2-cell embedded on a surface S. However Fréchet and Fan's book [13] gives priority of this formula for the sphere to Descartes. According to [17], Lhuilier generalized the formula to all orientable surfaces. The following theorem for planar graphs is well known.

Theorem 2.3 (Kuratowski, [25]) *A graph G is non-planar if and only if it contains a subdivision of $K_{3,3}$ or K_5.*

Kuratowski's Theorem was proved independently and published in 1927 by Pontryagin (see [27]). The theorem implies that the problem of recognizing if a graph is planar or not can be resolved in polynomial time (for example, see [22]). Hopcroft and Tarjan [19] were the first to devise a practical linear time algorithm to check if a graph G is planar or not. If G is a non-trivial 2-connected planar graph having at least four vertices, then a planar rotation system for G can always be transformed into a 2-cell toroidal or projective planar rotation system. Several methods to do this transformation are described in [15] and in Chapter 4 of this thesis. Some of the methods are implemented in the software *Groups&Graphs* [24].

There exist a number of linear time planarity testing algorithms described, for example, in [5], [19], [23] and [39]. In general, a planarity testing algorithm

can be modified so that in the case of a non-planar graph G it will return a subdivision of K_5 or $K_{3,3}$ in G (a subdivision of K_5 or $K_{3,3}$ exists in G by Theorem 2.3). A planar graph can be trivially embedded on the projective plane and torus. Therefore we can assume that G is a non-planar, 2-connected graph with no vertices of degree two (see previous remarks for graphs that are subdivisions or not 2-connected).

The projective plane and torus are the topological surfaces closest to the sphere in genus. Known embedding algorithms for these surfaces in [21], [28] and [30] begin with a subdivision of K_5 or $K_{3,3}$ in G, and try to extend an embedding of it to an embedding of G in the projective plane or torus. The results of this thesis and [14] simplify this approach by reducing projective planarity and toroidality for graphs with a K_5-subdivision either to planarity checks, or to the case of a $K_{3,3}$-subdivision in the graph.

Definition 2.9 To *contract* an edge $\{x, y\}$ of graph G means to remove the edge and both its endpoints x and y from G, and add a new vertex to G adjacent to all vertices in the neighborhood of x or y in G instead. The resulting graph is denoted by $G/\{x, y\}$. A graph H is a *minor* of G if H can be obtained from G by removing and/or contracting some subset of its edges, deleting resulting isolated vertices and identifying multiple edges. We write $H = MG$ to denote a minor H of G.

The following theorem is a generalization of Kuratowski's Theorem 2.3 and is known as *Wagner's theorem.*

Theorem 2.4 (Wagner's theorem, [8]) *A graph G is non-planar if and only if it contains $K_{3,3}$ or K_5 as a minor.*

Initially, a generalization of Kuratowski's theorem for non-orientable surfaces was described in [2]. For orientable surfaces, this was done in [3]. The following theorem is a generalization of Wagner's theorem for an arbitrary surface.

Theorem 2.5 (Robertson and Seymour, [35]) *For every surface S there exists a finite set of graphs $\{H_1, H_2, \ldots, H_n\}$ such that a graph G is non-embeddable in S if and only if it contains a minor from the set $\{H_1, H_2, \ldots, H_n\}$.*

The result of Theorem 2.5 is usually stated as a corollary to the *Graph Minor Theorem* as in [8]. The fact that there is a finite set of minors $H_1, H_2, \ldots, H_n$ for any surface S implies a polynomial time algorithm to determine if G is embeddable on any given surface S (see [28] and [29]). However the explicit list of minors is known only for the sphere [8] and projective plane [1]. There are tens of thousands of known minors for the torus (personal communication by W. Myrvold and [6]). References [21], [28] and [29] present linear time algorithms for the torus, projective plane and an arbitrary surface, respectively. There are currently no known implementations of these algorithms. So many

details have been omitted and the algorithms are sufficiently complex, that implementing them or checking their correctness are not easy. This is mentioned in the review [41]. The algorithms may be considered as theoretical models for graphs on surfaces. They provide some interesting insights into the algorithmic aspects of the problem as stated in [41].

Chapter 3

Planarity Testing

This chapter contains a description of the Hopcroft-Tarjan planarity testing algorithm. Some of the points and ideas of the algorithm are used or developed later for our new projective planarity and toroidality testing algorithms. A planarity testing algorithm is used in the new algorithms for graphs with K_5-subdivisions. The description of this chapter is based on the book of Reingold, Nievergelt and Deo [33] and the paper of W. Kocay [23]. The algorithm is very involved with details and references to other algorithms. Therefore we provide a description that reasonably highlights all the main points and ideas.

Notice that there are several versions of the Hocroft-Tarjan planarity algorithm presented in [19], [39], [23]. The algorithm can be modified to find a K_5-

subdivision or $K_{3,3}$-subdivision in a non-planar graph G. Such a procedure is described, for example, in [40] and is mentioned as an exercise in [33].

The algorithm of Boyer and Myrvold [5] is a simplification of known linear time planarity algorithms. However the description of [5] omits details for a better understanding of the algorithmic correctness.

A very simple planarity testing algorithm is described by Klotz in [22]. The algorithm is based on a constructive proof of Kuratowski's Theorem 2.3. However its time complexity is $O(n^2)$.

Definition 3.1 A graph G is *planar* if it is embeddable in the plane.

As it is stated in [33], *"the problem of determining whether a graph can be drawn on a plane without any edges crossing is of great practical interest"*. The problem is different from other graph-theoretical problems in that a graph drawn on a surface has an interplay between continuous properties of the surface and discrete properties of the graph. The characterization of Theorem 2.3 provides a classical example of this interplay.

The main idea of the algorithm presented here consists of considering consecutively bigger planar subgraphs of a graph and trying to complete them to a planar embedding of the whole graph. Clearly, a simple graph G is planar if and only if a directed graph obtained by orienting edges of G is planar. Also G is planar if and only if all its connected components are planar. Finally, it is

easy to see that G is planar if and only if all its 2-connected components called *blocks* are planar (for example, see [27]). So, for the algorithm, we consider simple 2-connected graphs.

For a connected graph $G = (V, E)$ embedded in the plane, Euler's formula relating the number of vertices $|V| = n$, edges $|E| = m$ and faces f of the embedding is

$$f + n = m + 2.$$

The formula implies that a graph G with $n > 2$ and $m > 3n - 6$ can never be planar (for a proof see [27]). A graph G is *complete* if every pair of distinct vertices is an edge in G. The complete graph on four vertices, K_4, is clearly planar. Therefore all subgraphs of K_4 are planar. By Theorem 2.3, a minimal non-planar graph has at least 9 edges. Therefore, the planar cases of $n < 5$ or $m < 9$ can be described explicitly, and we can assume that graph G is simple, 2-connected, $n \geq 5$ and $9 \leq m \leq 3n - 6$.

Initially the Hopcroft-Tarjan algorithm finds a cycle C in G and embeds C as a simple closed curve in the plane. By Jordan's Theorem, it divides the plane into two separate regions: the interior and exterior of C. Denote the edges of C by EC. Then the algorithm decomposes $G \backslash EC$ into edge-disjoint paths and tries to embed each path entirely either in the interior of C or in the exterior of C. If it succeeds in embedding the entire graph G in this way, then G is planar. Otherwise G is non-planar and it is possible to modify the algorithm

to find a subdivision of K_5 or $K_{3,3}$ (for example, see [40]) which exists by Kuratowski's Theorem. The difficulty of this approach is in embedding the paths of $G \backslash EC$. It must be guaranteed that an initial wrong embedding of a path does not lead us to an incorrect conclusion that G is non-planar when G still admits a planar embedding. So, the Hopcroft-Tarjan algorithm considers the paths in an appropriate order to choose the right place to embed them and, possibly, rearranges embedded paths to properly add the remaining ones.

3.1 Cycle and Paths Decomposition

The Hopcroft-Tarjan algorithm considers the graph as decomposed into a cycle and a set of edge disjoint paths. Initially a depth-first search is run on the graph to obtain a DFS-numbering of its vertices. A DFS-numbering corresponds to the order in which the vertices are visited by a depth-first search. Then we need to reorder the vertices and the adjacency lists of G according to the DFS-numbering. This is done to ensure the algorithm's correctness.

First we run a depth-first search algorithm on G to obtain a DFS-numbering of G (for example, see [33] or [27]). Also the DFS algorithm constructs a spanning tree of G called a *DFS-tree*. It is convenient to consider graph G as converted into a digraph G' according to the DFS-tree labelling. The DFS partitions the m edges of G into $n-1$ spanning *tree edges* and $m-n+1$ *back*

edges or *fronds* not in the DFS-tree. Now we can refer to a vertex v of G by its DFS-number. The DFS-tree edges are oriented from a smaller label vertex towards a bigger label vertex. By the DFS properties, a back edge $e = ab$ is always directed from a vertex a of higher DFS-number to a vertex b of lower DFS-number of a DFS-tree such that vertex b is an ancestor of a in the tree. An example of a DFS-tree and a digraph G' is shown in Figure 3.1. Trees will be drawn growing upwards from their root vertex.

Let $lowpt(v)$ (see [33]) be the lowest numbered vertex reachable from vertex v or from any of its descendants in the DFS-tree by means of at most one back edge. When it is not possible to reach a vertex below v by means of a single back edge from a descendant, v itself becomes $lowpt(v)$. Similarly, let $nextlowpt(v)$ be the next lowest vertex below v, excluding $lowpt(v)$, that can be reached in the same way. If there is no such vertex, $nextlowpt(v)$ is equal to v.

If we denote by S_v the set of all vertices lying on any directed path from vertex v consisting of zero or more DFS-tree edges and ending in at most one back edge of G', then we have the following formal definition.

Definition 3.2 The *low point* of vertex v in G', denoted by $lowpt(v)$, is

$$lowpt(v) = min(S_v)$$

and the *second low point* of v in G', denoted by $nextlowpt(v)$, is

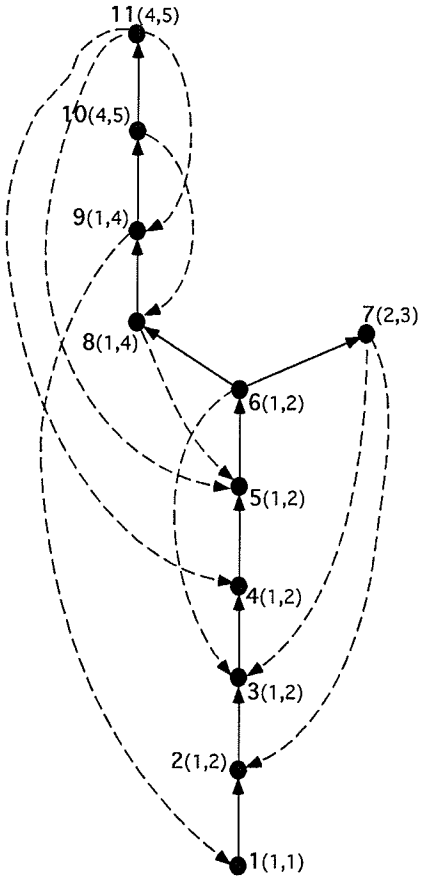

Figure 3.1: DFS-tree and digraph G' with low points

$$nextlowpt(v) = min(\{v\} \cup (S_v - \{lowpt(v)\})).$$

For example, for vertex 8 in Figure 3.1, $lowpt(8) = 1$ and $nextlowpt(8) = 4$.

The root vertex of the DFS-tree is labelled 1. Since graph G is 2-connected, we can always reach a vertex lower than v by means of paths ordered from v and ending by a back edge except when v is the root of the DFS-tree. This implies that for $v \neq 1$, $v \geq nextlowpt(v) > lowpt(v)$, and for the root $v = 1 = lowpt(v) = nextlowpt(v)$.

The low points are used to re-order the adjacency lists of G' so that during a second depth-first search on G', the paths in G' are generated in a certain necessary order. To do this efficiently, it is necessary to use the following function on directed edges of G'.

Definition 3.3 For each directed edge ab of G', its *weight* $\phi(ab)$ is defined as:

$$\phi(ab) = \begin{cases} 2 * lowpt(b), & \textit{if } ab \textit{ is a tree edge and } nextlowpt(b) \geq a \\ 2 * lowpt(b) + 1, & \textit{if } ab \textit{ is a tree edge and } nextlowpt(b) < a, \textit{ and} \\ 2 * b, & \textit{if } ab \textit{ is a back edge.} \end{cases}$$

Assuming the low point values are known, the weight function ϕ is easy to calculate. Then for each vertex a, the edges incident to a can be sorted into a non-decreasing order according to their weights. The sorting of the adjacency lists can be done in $O(n+m)$ time by using well-known sorting algorithms (for

example, see [33]). Now we use this order for the adjacency lists of G' and call them *properly ordered.* In [23] the author uses a refinement of the reordered adjacency lists of G' and shows that it can be crucial to determine a planar graph.

Hereafter, we assume that the adjacency lists of G' are ordered according to the weight values. Notice that in the reordered adjacency list of any vertex u, a back edge uv always precedes a back edge uw if $v < w$. Also the tree edges uz are considered in a non-decreasing order according to their ability to lead to a vertex below u through a single back edge.

For example, the reordered adjacency lists of the DFS-numbered directed graph of Figure 3.1 are:

$1 \rightarrow (2);$
$2 \rightarrow (3);$
$3 \rightarrow (4);$
$4 \rightarrow (5);$
$5 \rightarrow (6);$
$6 \rightarrow (8, 7, 3);$
$7 \rightarrow (2, 3);$
$8 \rightarrow (9, 5);$
$9 \rightarrow (1, 10);$
$10 \rightarrow (11, 8);$

$11 \rightarrow (4, 5, 9)$.

Having obtained the ordered adjacency lists for the digraph G', another depth-first search can be used to decompose G' into a cycle C and a set of edge disjoint paths $\{p_1, p_2, \ldots, p_{m-n}\}$ of $G' \backslash EC$. The cycle C and each path of $\{p_1, p_2, \ldots, p_{m-n}\}$ is determined by and contains a back edge of G'. The pseudo-code presented in Algorithm 3.1 does the cycle and path decomposition of G' according to the reordered adjacency lists $AdjList(v)$ for vertices v of G'.

Algorithm 3.1 *Decomposes a DFS-digraph G' represented by properly ordered adjacency lists into a cycle p_0 and paths $p_1, p_2, \ldots, p_{m-n}$.*

Input: *A DFS-digraph G' represented by properly ordered adjacency lists.*

Output: *Cycle p_0 and edge disjoint paths $p_1, p_2, \ldots, p_{m-n}$ of G'.*

begin

1. $i = 0$
2. $p_i = \emptyset$
3. $Path(1)$

end

procedure $Path(a)$

  for each $b \in AdjList(a)$ considered in the order of $AdjList(a)$ **do**

    $p_i = p_i \cup \{ab\}$

    if $a < b$ **then** /* *ab is a tree edge* */

$Path(b)$

else /* *ab is a back edge* */

$i = i + 1$

$p_i = \emptyset$

end if-else

end for

return

Informally, starting at the root vertex 1, the initial cycle p_0 is obtained by adding consequently the first edge in the vertex adjacency list which is a DFS-tree edge until we encounter a vertex z such that the first edge in the reordered adjacency list of z is a back edge. The back edge from z goes back to the root vertex 1. This back edge together with the path of tree edges from 1 to z forms the initial cycle $C = p_0$.

Then we begin from z and start a new path p_1 with the next edge out of z (the first edge out of z is the back edge to the root 1). Each time that we traverse a tree edge, we continue building the current path. When we traverse a back edge, it becomes the last edge of the current path. Thus each path consists of a sequence of zero or more tree edges followed by a single back edge. A new path is started from the initial vertex of the last back edge. If this vertex has no more unexplored edges, we back up to the previous vertex on the last path. The process is continued until G' has no more untraversed edges.

For example, Algorithm 3.1 working on the reordered adjacency lists of the DFS-numbered directed graph G' of Figure 3.1 provides the following cycle and path decomposition of G':

$C = p_0 = (1, 2, 3, 4, 5, 6, 8, 9, 1)$;
$p_1 = (9, 10, 11, 4)$;
$p_2 = (11, 5)$;
$p_3 = (11, 9)$;
$p_4 = (10, 8)$;
$p_5 = (8, 5)$;
$p_6 = (6, 7, 2)$;
$p_7 = (7, 3)$;
$p_8 = (6.3)$.

3.2 Properties of the Path Decomposition

The initial DFS-numbering of G provides a unique decomposition of G' into the cycle and paths. In general, the decomposition of a given graph G into a cycle $C = p_0$ and a sequence of edge-disjoint paths $p_1, p_2, \ldots, p_{m-n}$ is not unique. However the number of paths is $m - n$, since the cycle and each path p_i, $i \geq 1$, contains exactly one back edge.

We describe some properties of the decomposition that are used for the planarity testing algorithm. The vertices of G have been relabelled by their DFS-numbers. Suppose the generated cycle C is $p_0 = (v_1, v_2, \ldots, v_k, v_1)$, where $v_1 = 1$ and $v_1 < v_2 < \ldots < v_k$. Every path p_i has only its end vertices in common with the union of previously generated cycle and paths $p_0 \cup p_1 \cup p_2 \cup \ldots \cup p_{i-1}$, $i \geq 1$. Only the two end vertices of each path are required to distinguish between paths for the algorithm. Therefore for a generated path p_i, $i \geq 1$, we denote its start vertex by s_i, its finish vertex as f_i and we refer to the path as $p_i = (s_i, f_i)$.

3.2.1 Path Properties

Each path generated by Algorithm 3.1 ends with a back edge. From all available back edges incident on a particular vertex, Algorithm 3.1 selects the back edge that leads to the vertex with the smallest depth-first number of those which has not been used before. The selection happens automatically because of the non-decreasing order of the adjacency lists with respect to the weight function ϕ.

Lemma 3.1 ([19]) *For a path $p_i = (s_i, f_i)$, f_i is the lowest vertex reachable from s_i by a sequence of tree edges and any one of the back edges that have not been used in any path when the first edge in path p_i is traversed. Furthermore, for an intermediate vertex v in path p_i, i.e. $v \in p_i$, $v \neq s_i$ and $v \neq f_i$,*

$f_i = lowpt(v)$.

Proof. The lemma follows from the fact that all edges from v and its descendants are untraversed when the first edge in p_i is taken, and from the re-ordering of the adjacency lists according to the weight function values. ■

Lemma 3.2 ([19]) *For two generated paths $p_i = (s_i, f_i)$ and $p_j = (s_j, f_j)$, $j > i \geq 1$, if s_i is an ancestor of s_j in the tree ($s_i \leq s_j$), then $f_i \leq f_j$.*

Proof. This is true because the back edge ending p_j was unused when p_i was being generated. By Lemma 3.1, p_i takes the back edge reaching the lowest possible vertex in G' at this time. ■

Lemma 3.3 ([19]) *For two generated paths $p_i = (s_i, f_i)$ and $p_j = (s_j, f_j)$, $j > i \geq 1$, such that $s_i = s_j = s$, $f_i = f_j = f$, x_i is the second vertex in path p_i and x_j is the second vertex in path p_j, if edge sx_i is not a back edge and $nextlowpt(x_i) < s$, then sx_j is not a back edge and $nextlowpt(x_j) < s$.*

Proof. By Lemma 3.1, since $x_i \neq f$, $f = lowpt(x_i)$. Since sx_i is a tree edge and $nextlowpt(x_i) < s$, by definition, the weight function value of the edge is

$$\phi(sx_i) = 2lowpt(x_i) + 1 = 2f + 1.$$

Since p_j is generated after p_i, vertex x_j must appear later than x_i in the properly ordered adjacency list $AdjList(s)$. Therefore we have

$$\phi(sx_j) \geq \phi(sx_i) = 2f + 1.$$

Thus sx_j is not a back edge either and $x_j \neq f$. Also, $nextlowpt(x_j) < s$, otherwise we would have

$$\phi(sx_j) = 2lowpt(x_j) = 2f,$$

a contradiction. ■

Notice that Lemma 3.3 uses the *nextlowpt* values. We will need the *nextlowpt* values to break a tie between two tree edges uv and uw, directed from a common vertex u and having $lowpt(v) = lowpt(w)$ in G'. Lemmas 3.1-3.3 and their proof provide insight into a correct working of the planarity algorithm.

3.2.2 Segments

If we remove edges of the cycle $C = p_0$ from G', the subgraph induced by the edges of the remaining digraph $G' \backslash EC$ partitions the vertex set of G into connected components. Each connected component of $G' \backslash EC$ consists of one or more segments defined as follows.

Definition 3.4 A *segment* S of G' with respect to the cycle C is either a single back edge $v_i w$ such that $v_i w \notin C$, but $v_i \in C$ and $w \in C$, or a subgraph consisting of a tree edge $v_i w$, $v_i \in C$, $w \notin C$, and the directed subtree rooted

at w, together with all back edges from this subtree. The vertex $v_i \in C$ at which segment S originates is called the *base vertex* of S.

Each segment S is a connected subgraph of $G'\backslash EC$, but not every connected subgraph of $G'\backslash EC$ is a segment. For example, several segments may have a common base vertex. Algorithm 3.1 generates segments in a decreasing order according to the depth-first numbers of the base vertices. Moreover, all paths of a segment are generated before any path of the next segment. Clearly, all paths of the same segment must be embedded together either inside or outside of the cycle C. This is a reason to group paths into the segments and to consider them in segments.

For example, the DFS-numbered directed graph G' of Figure 3.1 has a decomposition into the cycle $C = p_0$ and four following segments S_1, S_2, S_3 and S_4 (see Section 3.1 for an explicit description of p_0 and the paths $p_1, p_2, \ldots, p_8$):

$S_1 = \{p_1, p_2, p_3, p_4\}$;
$S_2 = \{p_5\}$;
$S_3 = \{p_6, p_7\}$;
$S_4 = \{p_8\}$.

While embedding a segment, the Hopcroft-Tarjan algorithm can be applied recursively and can generate segments inside of a segment with respect to

another cycle. This is another main idea of the algorithm.

3.3 Main Features of the Embedding Algorithm

The cycle C can be embedded either clockwise or counter-clockwise in the plane as a simple closed curve. Without loss of generality, we assume that C is embedded counterclockwise in the plane and we need to add the segments to the embedding of C without any edges crossing. Consider a segment S with the base vertex v_i and the first edge v_iw.

Definition 3.5 Segment S is said to be embedded *inside of* C, if the edges incident on the base vertex v_i appear in the adjacency list of v_i clockwise as $v_{i-1}v_i$, v_iw, v_iv_{i+1}. Otherwise S is embedded *outside of* C and the edges incident on the base vertex v_i appear in the adjacency list clockwise as $v_{i-1}v_i$, v_iv_{i+1}, v_iw, where $i-1$ and $i+1$ are taken modulo k. A back edge xv_j, $v_j \in C$, belonging to a segment embedded on the inside of C, is said to be *entering* C *from inside.* Otherwise the segment is embedded on the outside of C and its back edge xv_j, $v_j \in C$, is *entering* C *from outside.*

3.3.1 Embedding the First Path of a Segment

The segments of $G'\backslash EC$ are generated by Algorithm 3.1 in a certain order. We are trying to embed the segments of $G'\backslash EC$ one at a time in the same order. To embed a segment S, we consider the first path p of S generated by Algorithm 3.1. Without loss of generality, we are trying to embed p inside of C by examining the previously embedded paths in constant time as follows. If p can be embedded inside of C, we embed it there. Otherwise all the previously embedded segments that are blocking the embedding of p inside of C are moved to the outside of C. Moving the segments from inside to outside of C may force some other segments to be moved from outside to inside of C, etc. If p cannot be embedded inside of C after the rearrangement of segments, then graph G is non-planar. If p can be embedded inside of C, we embed it there and try to embed the remaining paths of segment S by applying the embedding algorithm recursively. When we successfully embed S, we continue with the next segment in the same way.

The following theorem is used to check efficiently if the first path p of segment S can be embedded on a specific side of C during the embedding algorithm. Suppose $p = (v_i, v_j)$, where $v_i, v_j \in C$, $v_j < v_i$, and all the segments generated before S have been successfully embedded.

Theorem 3.1 ([33]) *Path p can be embedded inside of C, if there is no previously embedded back edge xv_t, $v_t \in C$, entering C from inside such that*

$v_j < v_t < v_i$. *Furthermore, if there is such a back edge* xv_t, *then* S *cannot be embedded inside of* C.

Proof. The segments are generated in the decreasing order of the depth-first labels of their base vertices. Therefore none of the edges embedded so far can be leaving any of the vertices smaller than v_i in C. Thus if no back edges enter C from inside between v_i and v_j, nothing prevents us from embedding p inside of C by placing p sufficiently close to C.

Suppose that there is an embedded back edge xv_t entering C from inside such that $v_j < v_t < v_i$. Back edge xv_t belongs to a previously generated segment S' embedded inside of C. Denote by v_q the base vertex of segment S'. Notice, that if S' is a single back edge xv_t, then $v_q = x$. Since S' was generated before S, then $v_q \geq v_i$. If $v_q > v_i$, then the sequence of edges in S' from v_q to v_t would cross an embedding of p inside of C and we cannot embed p inside of C without edges crossing.

Therefore we assume that $v_q = v_i$. It means that segments S and S' have a common base vertex. We consider the first path $p' = (v_q, v_r)$, $v_r \in C$, of S' generated by Algorithm 3.1. Lemma 3.2 implies $v_r \leq v_j$. Thus vertex $v_r \neq v_t$ and there are at least two paths in S', one entering C at v_t and the other at v_r. Also it means that the first path p' of S' is not just a back edge.

If $v_r < v_j$, then it is clear that path p cannot be embedded inside of C without

edges crossing. The last possibility is that $v_r = v_j$ and paths p and p' have both end vertices in common, i.e. $v_i = v_q$ and $v_j = v_r$. Let w and w' be the second vertices on paths p and p' respectively. Since the first edge of p' is not a back edge, $w' \neq v_r$. Moreover, there are at least two back edges in S' entering C below v_q at v_t and v_r. This implies $nextlowpt(w') < v_q$. This gives $w \neq v_j$ and $v_j < nextlowpt(w) < v_i$ by Lemma 3.3, i.e. there are at least two back edges in S going to v_j and to $nextlowpt(w)$ that lies between v_j and v_i. The sequence of edges from v_i to $nextlowpt(w)$ in S together with path p cannot be embedded inside of C without crossing an edge of S'. Therefore S cannot be embedded inside of C without edges crossing. ■

By Theorem 3.1, we need to know just the start and end vertices of a path to check its embeddability.

3.3.2 Recursion

Denote by Ep the set of edges of the path p. Having embedded the first path $p = (s, f)$ of a segment S, it is necessary to decide if $S \backslash Ep$ can be added to the planar embedding.

Lemma 3.4 ([33]) *$S \backslash Ep$ can be added to the planar embedding constructed before if and only if the subgraph $S \cup C$ is planar.*

Proof. By Theorem 3.1, there are no previously embedded back edges entering C from inside in the interval from f to s. On the other hand, all the back edges of $S \backslash Ep$ are entering p or C in the interval from f to s. The lemma follows. ■

Therefore the next step is to decide on the planarity of the subgraph $S \cup C$ of G'. This can be accomplished by applying Theorem 3.1 recursively to $S \cup C$. For the recursion, the DFS-tree edges from f to s on C and the first path p of S serve as the initial cycle C' for $S \cup C$. Then removing the cycle C' from $S \cup C$ may further partition the remaining digraph $(S \cup C) \backslash EC'$ into segments of $S \cup C$ with respect to the cycle C'. We can apply recursion to the segments of $S \cup C$ by considering them one by one as well. We continue the recursive process for segments and a cycle till all the paths of segment S are embedded in the plane or some path cannot be added to the embedding by Theorem 3.1.

3.3.3 Data Structures

For the segment embedding we consider them in a non-increasing order of the DFS-numbers of their base vertices on the cycle C. Suppose we are embedding a segment S with base vertex $s \in C$. To decide on the embeddability of S, we need to know which vertices of C smaller than s already have back edges entering from either the inside or the outside. Knowing these vertices of C, we can tell if we can embed S by using Theorem 3.1. Therefore we will use two

stacks *InsideStack* and *OutsideStack* to list vertices of C smaller than the current base vertex and having back edges entering C from the inside and the outside respectively. The stacks contain the vertices in increasing order with the biggest DFS-number vertex on top.

It is always attempted to embed a new segment inside of C first. Therefore, when the first path $p = (s, f)$ of S is generated, vertex $f \neq 1$ must be added on top of *InsideStack*. During the recursion, all the end vertices of the paths of S are added to *InsideStack* as well. Clearly, a new vertex is added to *InsideStack* if and only if its DFS-number is bigger than the top vertex in the stack. Otherwise the DFS-number of the vertex is the same as the top one and is already represented in the stack.

The segments are generated in a non-increasing order of the DFS-numbers of their base vertices on C. When we are moving down a tree edge $v_{i-1}v_i$ during the generation and embedding of the segments, we have all the segments with base vertices higher than v_{i-1} successfully embedded. Thus, by Theorem 3.1, the back edges entering cycle C at vertex v_{i-1} or higher do not interfere with the paths of the remaining segments. Therefore we can remove vertex v_{i-1} from *InsideStack* and *OutsideStack* as we move down from v_i to v_{i-1} on the edge $v_{i-1}v_i$.

The segments may be moved from inside to outside of C or vice versa several times. Moving one segment can initiate moving other segments. Clearly, the

corresponding entries of *InsideStack* and *OutsideStack* must be shifted as well. It is necessary to group back edges which are moving simultaneously as a bundle.

Definition 3.6 A *bundle* is a maximal set of entries in *InsideStack* and *OutsideStack* that corresponds to back edges such that an embedding of one of the back edges determines the embedding of all others.

When the stacks change, the set of bundles changes as well. However the bundles always partition the entries of both stacks. Furthermore, from the order of path generation and adding the new entries on top of a stack, all entries in each stack corresponding to the same bundle are adjacent.

Consider the embedding of a new segment S with the first path $p = (s, f)$ inside of C. When all entries corresponding to back edges in S are added to *InsideStack*, a new bundle B is formed. B contains the *InsideStack* entries for the segment S as well as any entry for a back edge that interferes with S, i.e. a vertex v such that $f < v < s$. Therefore every bundle that contains such a vertex v must be combined with the new entries of S to form the new bundle B. The vertices v' in all other bundles must satisfy $v' \leq f$. Some members of B will be in *InsideStack*, others will be in *OutsideStack*, but they all will be on top in both stacks. Since an embedding of a back edge in a bundle completely determines just the embedding of all other edges of the same bundle, it is useful to keep the back edges in bundles.

The information about bundles can be stored in a third stack *BundleStack*. Each entry in *BundleStack* is an ordered pair (x, y) for a bundle, where x is the lowest entry in *InsideStack* and y is the lowest entry in *OutsideStack* for the bundle. $x = 0$ means the bundle does not have any entry in *InsideStack* and $y = 0$ means the bundle does not have any entry in *OutsideStack*. It is convenient to implement entries (x, y) of *BundleStack* as pointers to the corresponding entries of *InsideStack* and *OutsideStack*.

This presentation of the algorithm is taken from [33]. An actual embedding algorithm is developed inside of the path decomposition Algorithm 3.1. Its complete detailed description is provided in [33].

3.3.4 Complexity of the Algorithm

The DFS path generation part of the algorithm requires $O(n+m)$ operations. The embedding part uses information about $m-n$ endpoints of the generated paths. It consists only of a sequence of stack manipulations, and adding or deleting an element requires a constant time. The total number of entries in the stacks is $O(n+m)$. Therefore the entire algorithm runs in $O(n+m)$ time. Since $m \leq 3n - 6$ for a planar graph, the algorithm requires $O(n)$ operations. The storage memory use is also $O(n)$. A more subtle analysis of the algorithm and its complexity can be found in [23].

The algorithm can be modified to obtain a rotation system for an actual embedding of the graph by using information about the stacks and their changes. Also it can be modified to return a subdivision of K_5 or $K_{3,3}$ in G when the graph is non-planar. A description in [33], [23] and [40] provides more detailed ideas to do it in linear time as well.

3.4 Summary of the Hopcroft-Tarjan Planarity Algorithm

In this section we consider the Hopcroft-Tarjan planarity algorithm in a simplified intuitive form. The simplified ideas are used later for the projective planarity algorithm for graphs with a $K_{3,3}$-subdivision.

3.4.1 A Spanning Initial Cycle

Suppose we first consider a graph G with a spanning DFS-tree that is a path and the spanning cycle $C = p_0$ is the starting subgraph for an embedding of G. The cycle C divides the plane into 2 regions. Each back edge is a chord of the cycle. We then have to place the remaining chords of G either inside or outside the cycle.

If G has n vertices, the vertices of the cycle are then DFS-numbered $1, 2, ..., n$ along the cycle. We sort the adjacency lists in order of increasing DFS-number of adjacent vertices.

To embed the chords, first we start at vertex numbered 1, and travel along the cycle to the last vertex n as the first DFS algorithm does. We then consider all chords at n. Since they are ordered by their endpoints, we can easily place them inside C. Then we move back to vertex $n-1$ and place the chords whose higher endpoint is $n-1$. It is easy to tell if a chord can be embedded on the inside of C, because the cycle is ordered by the DFS-numbers. We just compare endpoints. If a chord does not embed on the inside, we embed it on the outside. If a chord cannot be embedded on either side, it may be possible to switch a bundle of chords from the inside to the outside, and vice versa, as shown in Figure 3.2.

When we return to vertex 1 on the cycle, all the chords are embedded, and it provides a planar embedding of G. Instead of moving forward and backward to traverse the cycle C, it is possible to consider starting at the vertex labelled n, and simply following the cycle back to vertex 1. A similar technique is used in the projective planarity algorithm to traverse the boundaries of the faces of an embedding of $K_{3,3}$.

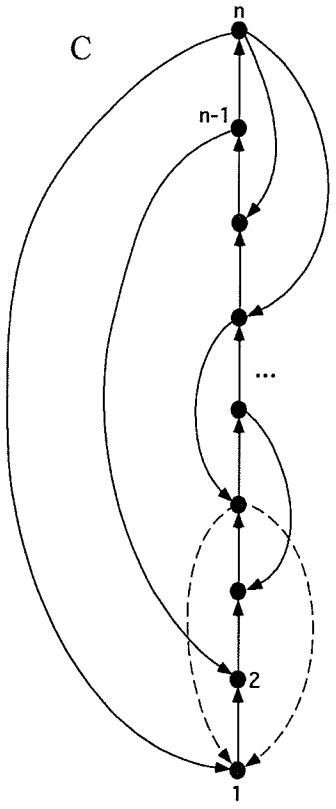

Figure 3.2: Embedding chords around a DFS-cycle C

3.4.2 Recursion for Traversing a Non-Spanning Initial Cycle

Now, if $C = p_0$ is not a spanning cycle, consider a vertex $v \in C$ visited by the algorithm as we move backward on the cycle C. If v is adjacent to some vertex $u \notin C$, there will be a path p' beginning with the edge vu, and meeting C in another vertex $w \in C$ (w is the end vertex of p'). Then path p_C from w to v on C and path p' together form a cycle $C' = p_C \cup p'$. We can apply the same approach recursively to embed the cycle C' and all its back edges.

Each back edge of C' is either placed inside C', or outside C'. The path p' itself is either placed inside C, or outside C. In general, we use recursively a depth-first search algorithm $DFS(u)$ for each vertex u adjacent to v (each vertex of the graph is visited just once). Thus, the Hopcroft-Tarjan algorithm is the recursive generalization of the algorithm which places chords either inside or outside of a cycle C. Its outline with the pseudo-code for the recursive *EmbedBranch* procedure is given in Algorithm 3.2. A detailed pseudo-code for the subroutine $SwitchSides(v, u)$ is provided in [23]. Similar detailed ideas are described in [33].

Algorithm 3.2 *The Hopcroft-Tarjan Planarity Testing.*

Input: *A graph G represented by adjacency lists*

Output: *G is planar or non-planar*

(1) Choose a starting vertex v of G

(2) Run $LowPointDFS(v)$

/ The procedure assigns DFS-numbers, calculates LowPoints, and orders the adjacency lists. We have now constructed a DFS-tree T that will be used to order the embedding of back edges. */*

(3) Run $EmbedBranch(v)$

/ We must traverse the DFS-tree T to its first leaf and find the initial cycle C. We then back up the tree as the recursion unfolds, and call the EmbedBranch procedure recursively to embed each branch of the tree that has its root on the cycle C. Each branch of the tree T corresponds to a segment of the DFS-digraph G'. The entire algorithm can be written as a DFS procedure to traverse T and to embed back edges. The procedure EmbedBranch does a traversal of T which starts at the top of T, and then descends the DFS-tree T, embedding back edges as the recursion unwinds. The procedure is calling itself recursively for each branch of the tree.*/*

procedure $EmbedBranch(v)$

  for each u adjacent to v **do**

    if (u has not been visited before)

      $EmbedBranch(u)$

      / The first back edge encountered will be incident to a point visited*

before and having a smaller DFS-number in the tree. This creates a cycle C. Subsequent back edges and branches will be embedded either inside C, or outside C./*

if (NonPlanar) **return**

else

*/*vu is a back edge. Either u is above v in the tree, or below this can be detected by comparing DFS-numbers.*/*

if (u is above v in T) **return**; */*The adjacency lists are ordered.*/*

*/*Otherwise u is below v in the tree.*/*

if (DFS-number of u fits inside C)

place uv inside C

else if (DFS-number of u fits outside C)

place uv outside C

else

*/*uv does not fit inside or outside C.*

Try switching sides./*

$k = SwitchSides(v, u)$

if ($k = 0$)

*/*Switching sides does not help.*/*

$NonPlanar = true$

return

*/*Otherwise switching sides made it possible to embed uv.*/*

if ($k = 1$) place uv inside C

else place uv outside C

end if-else DFS-number of u fits

end if-else u visited

end for

end *EmbedBranch*

A description of how to store the back edges uv in the linked lists *InsideStack* and *OutsideStack*, and how to switch sides using *BundleStack* for the inside and outside of the cycle are presented in Section 3.3.3. The details for an implementation of the stacks as linked lists are given in [33].

Chapter 4

Essential Cycles on the Projective Plane and Torus

This chapter presents graphs embedded on surfaces of higher genus than the plane. Let $G = (V, E)$ be a 2-connected graph, $|V| = n, |E| = m$. We are interested in 2-cell embeddings of graphs on the projective plane and torus. An embedding of G on a surface that is not a 2-cell embedding does not fit the surface and can be considered as an embedding on a surface of smaller genus. Let C be a cycle in graph G.

Definition 4.1 A cycle C is *contractible* or *null homotopic* with respect to an embedding of G on a surface if C can be contracted continuously on the

surface into a point. Otherwise C is an *essential* cycle of the embedding.

A spanning tree T of graph G has $n-1$ edges. Each of the remaining $m-n+1$ edges $e_1, e_2, \ldots, e_{m-n+1}$ of $G \backslash ET$ determines a unique cycle C_i in $T \cup e_i$, $i = 1, 2, \ldots, m-n+1$, called the *fundamental cycle* of e_i with respect to T. The fundamental cycles $\{C_1, C_2, \ldots, C_{m-n+1}\}$ are pairwise distinct and any closed circuit of G can be uniquely represented as a linear combination of the cycles $C_1, C_2, \ldots, C_{m-n+1}$ with respect to exclusive OR operation on the edges of two cycles. The set $\{C \mid C$ is a closed circuit in $G\}$ is a vector space over $GF(2)$ with basis $\{C_1, C_2, \ldots, C_{m-n+1}\}$. A good description of fundamental cycles and circuit spaces of a graph is provided in [16].

Definition 4.2 ([16]) The set of fundamental cycles $\{C_1, C_2, \ldots, C_{m-n+1}\}$ corresponding to a spanning tree T of graph G is called a *cycle basis* of G with respect to T.

4.1 Embedding Cycles on the Projective Plane

According to Chapter 2 and [13], we consider the projective plane as a disk with antipodal boundary points identified. For any 2-cell embedding of graph G on the projective plane, there is a cycle $C \in G$ that is essential for the embedding (see Figure 4.1). Otherwise G is planar embedded in the projective plane and

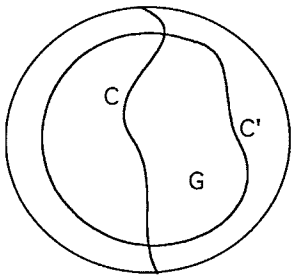

Figure 4.1: Essential and contractible cycles on the projective plane

one face of the embedding is not equivalent to an open disk. If we cut the projective plane along an essential cycle, we obtain a unique face equivalent to an open disk (the disk boundary consists of two copies of the cycle). Denote by F the face obtained by cutting the projective plane along the essential cycle C. $G\backslash C$ is then planar embedded in F. However the cycle C appears twice on the boundary of F and initially we do not know which copy of C on the facial boundary is adjacent to a vertex of $G\backslash C$.

Theorem 4.1 ([34]) *For a non-planar graph G embedded on the projective plane, there is a cycle C of any cycle basis $\{C_1, C_2, \ldots, C_{m-n+1}\}$ of G such that C is essential in the projective planar embedding of G.*

Proof. Suppose all the cycles of a cycle basis $\{C_1, C_2, \ldots, C_{m-n+1}\}$ are contractible. A linear combination of contractible cycles is a contractible cycle. However, by Theorem 2.3, G contains a subgraph of TK_5 or $TK_{3,3}$ and any embedding of TK_5 or $TK_{3,3}$ on the projective plane contains an essential cycle

(see Figure 7.1 and Figure 7.2). A contradiction. ■

Corollary 4.1 ([34]) *Any cycle basis of TK_5 or $TK_{3,3}$ in G contains a cycle C that is essential in any given embedding of G on the projective plane.*

4.2 Embedding Cycles on the Torus

According to Chapter 2 and [13], the torus can be represented as a rectangle with opposite sides identified. Consider a 2-cell embedding of graph G on the torus. The embedding of G cuts the torus surface into open disk cells. Therefore there is a cycle C of G that is essential on the torus. The essential cycle C cuts the torus surface into a surface homeomorphic to a cylinder surface as in Figure 4.2. The cylindrical face is not equivalent to an open disk either. Therefore there must be a path P in G as in Figure 4.2 which cuts the cylindrical face into a 2-cell. All the faces of the embedding of G are obtained by cutting the 2-cell determined by the cycle C and the path P of G.

The 2-cell face can be drawn as a rectangle having two copies of C and P on opposite sides of its boundary as in Figure 4.3.

Definition 4.3 ([27]) A *θ-graph H* is a graph consisting of two vertices of degree 3 connected by three internally disjoint paths.

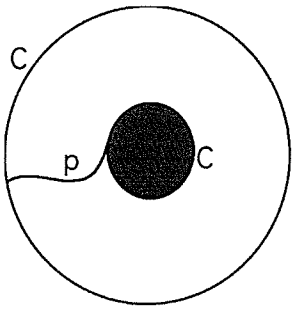

Figure 4.2: Cutting the torus surface into a 2-cell

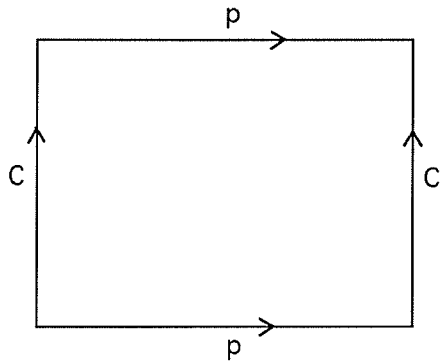

Figure 4.3: The torus cut into a 2-cell

If $P \cap C$ is a single vertex, then P is another essential cycle for the embedding of G and $C \cup P$ are two edge disjoint cycles sharing a unique common vertex. Otherwise $C \cup P$ contains two other different essential cycles for the torus embedding of G. The cycles share path P. In this case, $C \cup P$ is a θ-subgraph in G.

Theorem 4.2 ([34]) *For a non-planar toroidal graph G, at least two cycles of any cycle basis of G are essential in any embedding of G on the torus.*

By Theorem 2.3, a non-planar toroidal graph G contains a subdivision TK_5 or $TK_{3,3}$.

Corollary 4.2 ([34]) *At least two cycles of any cycle basis of TK_5 or $TK_{3,3}$ in G are essential in any embedding of G on the torus.*

4.3 2-Cell Embeddings of Planar Graphs on the Projective Plane and Torus

Consider a planar, 2-connected graph G that is not a cycle. We can embed G on the projective plane or torus in an arbitrary 2-cell disk. Clearly, a planar embedding of G on the projective plane or torus is not a 2-cell embedding.

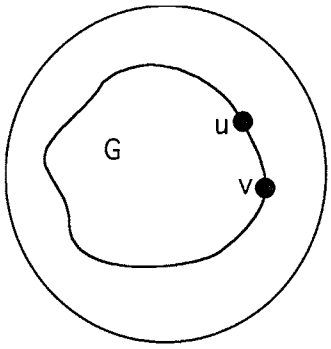

Figure 4.4: Planar embedding of G on the projective plane

However we can construct a 2-cell projective planar or toroidal embedding from a planar embedding of G as follows.

Suppose uv is an edge on the outer face boundary of a planar embedding of G, where G is embedded on the projective plane in an open disk as in Figure 4.4.

Then to transform the planar embedding of G into a 2-cell embedding of G on the projective plane we can just re-draw edge uv across the projective plane boundary as in Figure 4.5. Since G is 2-connected, edge uv will cross the projective plane boundary on an essential cycle for the embedding, creating a 2-cell embedding of G. In this case the rotation system of G does not change, but the edge uv has its signature changed from $+1$ to -1.

To obtain a toroidal 2-cell embedding of a planar graph G, we can use a θ-subgraph in G. A 2-connected graph G that is not a cycle must contain a

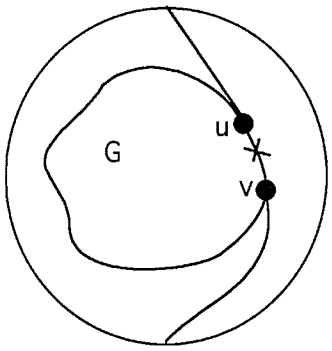

Figure 4.5: 2-cell embedding of planar G on the projective plane

θ-subgraph. To find a θ-subgraph H in G, we can either take the boundaries of two adjacent faces of a planar embedding of G or construct H by using a depth-first or breadth-first search. Denote by A and B the two vertices of degree 3 in H and by $P_1 = (A, a_1, \ldots, b_1, B)$, $P_2 = (A, a_2, \ldots, b_2, B)$, $P_3 = (A, a_3, \ldots, b_3, B)$ the three paths connecting A and B in H as in Figure 4.6.

The θ-subgraph H divides the plane into 3 regions. Without loss of generality, assume that path P_2 is inside the region bounded by $P_1 \cup P_3$ as in Figure 4.6. The region of the plane bounded by $P_1 \cup P_2$ contains a planar embedding of component C_3; the region bounded by $P_2 \cup P_3$ contains a planar embedding of component C_1; and the region bounded by $P_3 \cup P_1$ contains a planar embedding of component C_2 of $G \backslash H$ (see Figure 4.6). Then the adjacency list for vertex A in the planar embedding of G can be described as $A \rightarrow (a_1, v_1, \ldots, v_2, a_2, w_1, \ldots, w_2, a_3, u_1, \ldots, u_2)$, where $(v_1, \ldots, v_2)$ is the ordered neighborhood of A in C_3, $(w_1, \ldots, w_2)$ is the ordered neighborhood of A

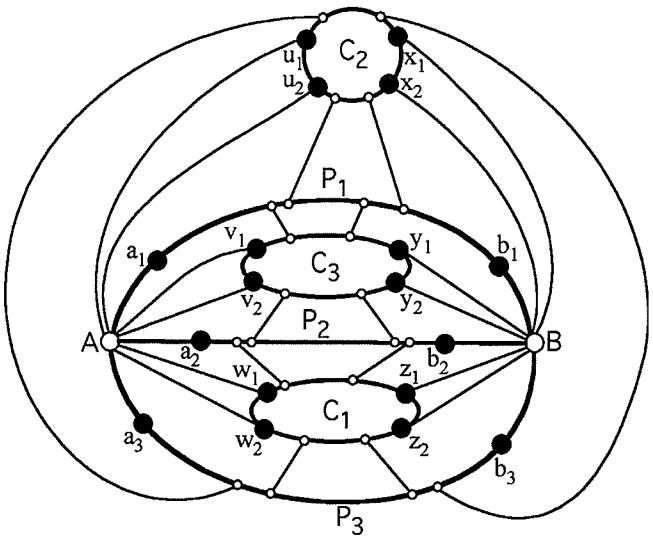

Figure 4.6: θ-subgraph in a planar embedding of G

in C_1 and $(u_1, \ldots, u_2)$ is the ordered neighborhood of A in C_2 (see Figure 4.6). Similarly, the adjacency list for vertex B in the planar embedding of G can be described as $B \rightarrow (b_1, x_2, \ldots, x_1, b_3, z_2, \ldots, z_1, b_2, y_2, \ldots, y_1)$ (see Figure 4.6).

A graph G with a rotation system r is denoted by G^r. Suppose G is planar with a θ-subgraph H as described above. Let G^p be the graph where p means a rotation system for its planar embedding. We can construct a rotation system G^t for a toroidal 2-cell embedding of G as follows.

Theorem 4.3 *Changing the adjacency list for vertex A of H from $A \rightarrow (a_1, v_1, \ldots, v_2, a_2, w_1, \ldots, w_2, a_3, u_1, \ldots, u_2)$ to*

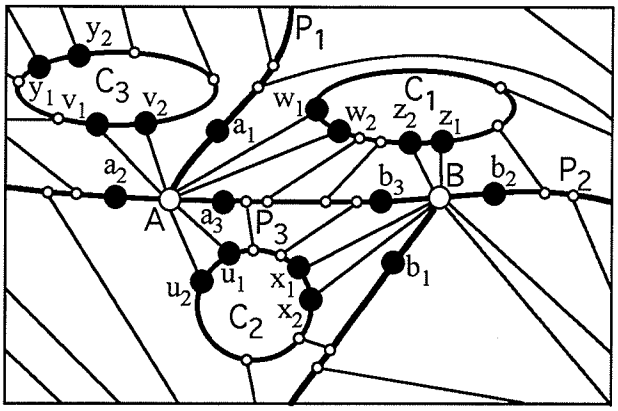

Figure 4.7: Converting a planar embedding into a 2-cell toroidal

$A \rightarrow (a_2, v_1, \ldots, v_2, a_1, w_1, \ldots, w_2, a_3, u_1, \ldots, u_2)$
in the planar rotation system G^p *provides a rotation system* G^t *for a toroidal 2-cell embedding of* G.

Proof. Notice that we change the adjacency list of A by altering the positions of just two vertices a_1 and a_2 corresponding to paths P_1 and P_2. This gives a 2-cell embedding of the θ-subgraph H on the torus in which the 3 paths P_1, P_2 and P_3 cut the torus surface into a single 2-cell as in Figure 4.7. We embed the remaining induced subgraphs C_1, C_2 and C_3 in the torus with the same rotation systems as in the plane. We just need to check that the connections of C_1, C_2 and C_3 to the vertices of H are in the same cyclic order in the torus as in the plane. This can be verified directly from Figure 4.7. Since the θ-subgraph H is 2-cell embedded on the torus, so is the entire graph G. ■

Theorem 4.3 provides a convenient way to convert a planar embedding of G into a 2-cell embedding of G on the torus. It requires interchanging just two edges in the adjacency list of one vertex. There are other methods that can be used to transform the planar rotation system G^p into a toroidal rotation system G^t.

Chapter 5

Graph Embedding Algorithms Currently Implemented for the Projective Plane and Torus

This chapter contains a description of the implemented algorithms of [30] and [31] for the projective plane and torus. We outline the main ideas of the algorithms that are important for better understanding of the corresponding surfaces, the data structures used by a computer program and the time complexity. The linear time algorithms of [21], [28] and [29] are more of a theoretical interest. We do not know of any linear time implementations of them.

5.1 Practical Projective Planarity Testing

The algorithm of [30] tries to complete an embedding of a subdivision TK_5 or $TK_{3,3}$ to an embedding of a graph G, as does the algorithm of [28].

Definition 5.1 A *bridge* of a graph G with respect to an embedded subgraph H is a subgraph B of G such that B is either an edge $\{v, w\} \in G \backslash EH$ with $v, w \in H$, or a connected component $C \in G \backslash H$ plus all edges $\{v, w\}$ such that $v \in C$ and $w \in H$. The vertices of $B \cap H$ are called *attachment points* of the bridge.

The algorithm in [30] is based on the idea of reduction to a 2-SAT problem. The 2-SAT problem is used to select an assignment of bridges to faces of an embedded $TK_{3,3}$ or TK_5. This is possible because an embedding of TK_5 or $TK_{3,3}$ on the projective plane (see Figure 5.1 and Figure 5.2) can be completed to a graph G by adding each bridge into at most three faces of the embedding. There can be just a constant number of bridges which are embeddable into three faces. After a preliminary selection of all possible embeddings for these bridges, the remaining bridges can be added into at most two faces of the embedding. The 2-SAT problem is used to assign the remaining bridges to faces.

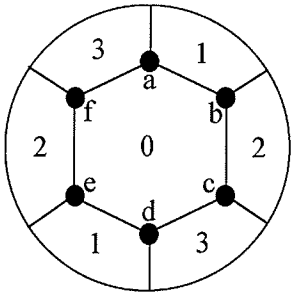

Figure 5.1: Embedding of $K_{3,3}$ on the projective plane

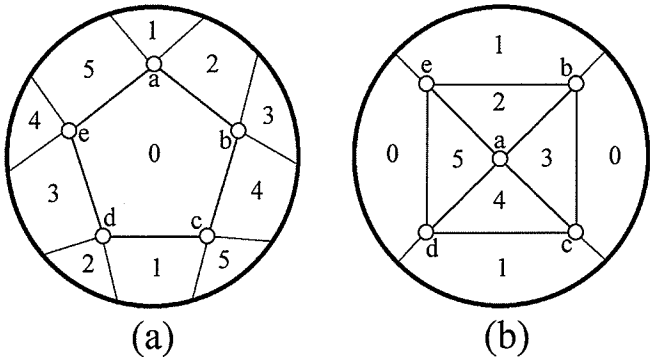

Figure 5.2: Embeddings of K_5 on the projective plane

5.1.1 Basic Ideas

By Theorem 2.3, a non-planar graph G contains a subdivision of K_5 or $K_{3,3}$. A modification of the planarity algorithm of Chapter 3 can return a subdivision TK_5 or $TK_{3,3}$ in G. We denote by TK a subdivision of K_5 or $K_{3,3}$ in G.

For each embedding of TK on the projective plane, the algorithm of [30] tries to embed the bridges of G with respect to TK in the faces of the embedding of TK. Each face of an embedding of TK is equivalent to an open disc (see Figure 5.1 and Figure 5.2). Therefore when a bridge is embedded into a face, the embedding of the bridge must be planar.

The algorithm in [30] considers all inequivalent embeddings of TK on the projective plane. By Definition 2.7, two embeddings are inequivalent if there is no such isomorphism between the corresponding rotation systems. Two embeddings with distinct face boundary walks have distinct rotation systems and therefore are inequivalent.

Theorem 5.1 ([21]) *$TK_{3,3}$ has one unlabelled embedding in the projective plane as in Figure 5.1 and TK_5 has two unlabelled embeddings in the projective plane as in Figure 5.2(a) and Figure 5.2(b). Furthermore, the number of inequivalent ways to label the embedding of $TK_{3,3}$ is* 6*, the number of inequivalent labellings of the embedding of Figure 5.2(a) is* 12*, and the number of inequivalent labellings of the embedding of Figure 5.2(b) is* 15.

A bridge B can be embedded in a face with boundary F if its attachment points are all in F and there is a planar embedding of $B \cup F$.

Definition 5.2 A bridge B is called a *k-face bridge* with respect to an embedding of TK if it can be embedded in k different faces of the embedding.

For any embedding of TK_5 or $TK_{3,3}$ in the projective plane, there can be 1-face, 2-face or 3-face bridges of G with respect to the embedding. The algorithm considers all the different assignments of the types of 3-face bridges to the faces of an embedding of TK. Then for each face of the embedding the algorithm determines the pairs of 1-face and 2-face bridges that cannot be embedded together in the face. This provides a 2-SAT problem to decide if there is an assignment of the bridges to the faces for an embedding of the whole graph G.

5.1.2 3-Face Bridges

For an embedding of the subgraph TK of G, there can be 3-face bridges of G. However it is possible to consider a small constant number of embeddings for the 3-face bridges to reduce the embedding problem to only 2-face and 1-face bridges. Here all the possibilities and combinations for embedding 3-face bridges are described. It is also explained how they are handled by the algorithm.

Any 3-face bridge B of a 2-connected graph G has just two attachment points in the vertices of TK corresponding to the vertices of K_5 or $K_{3,3}$. Referring to Figure 5.1 and Figure 5.2, the possible sets of the 3-face bridges according to their attachment points are:

for $TK_{3,3}$ of Figure 5.1:
$\{a, d\}$ for faces $0, 1, 3$, $\{b, e\}$ for faces $0, 1, 2$, $\{c, f\}$ for faces $0, 2, 3$;

for TK_5 of Figure 5.2(a):
$\{a, c\}$ for faces $0, 1, 5$, $\{a, d\}$ for faces $0, 1, 2$, $\{b, d\}$ for faces $0, 2, 3$, $\{b, e\}$ for faces $0, 3, 4$, $\{c, e\}$ for faces $0, 4, 5$;

for TK_5 of Figure 5.2(b):
$\{b, c\}$ for faces $0, 1, 3$, $\{b, e\}$ for faces $0, 1, 2$, $\{c, d\}$ for faces $0, 1, 4$, $\{d, e\}$ for faces $0, 1, 5$.

Without loss of generality, the 3-face bridges with the same set of attachment points can be treated as one 3-face bridge. Therefore the 3-face bridges are considered in groups corresponding to different pairs of attachment points. There is only a constant number of ways to assign the 3-face bridges to faces for an embedding of TK. It is necessary to consider all the combinations of embeddings for 3-face bridges to cover all possibilities and to leave at most two faces available for each remaining 3-face bridge. Then it is possible to apply the 2-SAT approach to embed the remaining bridges.

For the embedding of $TK_{3,3}$ in Figure 5.1, there is at most one group of 3-face bridges that can be embedded into face 0 without edges crossing. Thus, there are four ways to place 3-face bridges with respect to an embedding of $TK_{3,3}$: any of three groups of 3-face bridges can be embedded in face 0 or none of them is placed in face 0. In any case there are at most two remaining faces available to embed the 3-face bridges.

Consider the embedding of TK_5 of Figure 5.2(a). For all 3-face bridges, one of the possible faces to embed it is face 0. Thus, it is sufficient to choose the bridges that are placed in face 0 since there are only two choices for the other 3-face bridges. There are five different ways to simultaneously place two groups of 3-face bridges in face 0. Two groups must have a common corner as an attachment point. There are also five different ways to place just one group of 3-face bridges in face 0. Finally, there can be no 3-face bridges in face 0 at all. Thus, in total there are eleven cases to consider.

Consider the embedding of TK_5 of Figure 5.2(b). Bridges with attachment points $\{b, e\}$ and $\{c, d\}$ cannot simultaneously be embedded into face 0. Similarly, bridges with the attachment points $\{b, c\}$ and $\{d, e\}$ cannot simultaneously be embedded into face 1. Therefore there are nine different ways to assign 3-face bridges to faces so that face 0 contains either one group or none of 3-face bridges with attachment points $\{b, e\}$ and $\{c, d\}$, and face 1 contains either one group or none of 3-face bridges with attachment points $\{b, c\}$ and $\{d, e\}$. As a result, the remaining 3-face bridges have just two faces available

for their embedding. Thus, in total there are nine cases to consider.

To summarize the discussion, there can be at most three different groups of 3-face bridges for an embedding of $TK_{3,3}$, at most five different groups of 3-face bridges for an embedding of TK_5 of Figure 5.2(a), and at most four different groups of 3-face bridges for an embedding of TK_5 of Figure 5.2(b). The number of preliminary assigned embeddings of the 3-face bridges to faces is four for $TK_{3,3}$, eleven for TK_5 of Figure 5.2(a), and nine for TK_5 of Figure 5.2(b).

5.1.3 Conflicts Between Bridges and 2-SAT Problem

All the faces of a projective planar embedding of TK_5 or $TK_{3,3}$ are equivalent to an open disk and no vertex of TK_5 or $TK_{3,3}$ appears twice on a face boundary.

Definition 5.3 Two bridges B_i and B_j are *compatible* for a face if both can be embedded in the face simultaneously. Otherwise, they are called *conflicting* for the face.

Clearly, if bridges of a set are pairwise compatible for a face, then they can be embedded in the face simultaneously. We describe the ideas from [30] to determine conflicting pairs of bridges for a face. Since now we have just 1-face and 2-face bridges, the conflicting pairs provide us an instance of a 2-SAT

problem that can be resolved in linear time.

The number of bridges can be $O(n)$. Therefore the number of bridge pairs can be quadratic in n. Information about conflicting bridges can be computed in $O(n^2)$ time.

To find the pairs of conflicting bridges for a face, the vertices on the face boundary are considered in a cyclic order. In [30], a finite state machine (FSM) is used to determine if a pair of bridges (B_i, B_j) is conflicting. The FSM makes its transitions depending on whether the current face boundary vertex is an attachment point to B_i, B_j or both. The FSM computes the pairs of conflicting bridges in $O(n^2)$ time.

The remaining bridges are assigned to faces of an embedding of TK so that all the bridges embedded in the same face are compatible. If such an assignment exists, the input graph G is projective planar. The algorithm finds an assignment of the bridges to faces, or else it determines that no such assignment exists by formulating and resolving the corresponding 2-SAT problem.

Definition 5.4 ([30]) A *literal* is a boolean variable x or its complement $\bar{x}$. A set of literals is a *clause*. For a set of clauses, the *satisfiability problem (SAT)* is to determine whether there is an assignment of *true* and *false* values to the boolean variables so that at least one literal in each clause is *true*.

It is known that SAT is NP-complete [7] even when each clause contains at most three variables, i.e. 3-SAT. However if each clause has at most two variables, i.e. 2-SAT, the problem becomes linear and can be efficiently resolved by an algorithm like in [9].

Let F be a face of an embedding of TK and B be a bridge that can be embedded in face F. A literal B_F is assigned the value *true* if and only if bridge B is embedded in face F. Otherwise B_F is assigned the value *false*. If bridge B can only be embedded in face F, then a clause $\{B_F, B_F\}$ is added to the 2-SAT instance. This guarantees that B_F is assigned the value *true*. If bridge B is embeddable into two faces F_1 and F_2, two clauses $\{B_{F_1}, B_{F_2}\}$ and $\{\bar{B_{F_1}}, \bar{B_{F_2}}\}$ are added to the 2-SAT instance to guarantee that B is embedded in exactly one face. Finally, for each pair of bridges B_1 and B_2 conflicting in face F, a clause $\{\bar{B_{1F}}, \bar{B_{2F}}\}$ is added to the 2-SAT instance to guarantee that they both are not embedded in F simultaneously.

There can be $O(n)$ bridges, giving at most $O(n^2)$ conflicts between pairs of bridges. Thus it takes $O(n^2)$ time to resolve a 2-SAT instance using the algorithm of [9]. A solution to the 2-SAT instance provides an assignment of bridges to the faces of an embedding of TK that corresponds to an embedding of G.

5.1.4 Outline of the Quadratic Projective Planarity Algorithm

The algorithm of [30] runs on 2-connected graphs. If a connected graph G is not 2-connected, then it is necessary to find the blocks of G first. This can be done in $O(n)$ time by using a modified depth-first search as done in [33]. It is known that G is projective planar if it contains at most one projective planar block and all the other blocks are planar. Therefore, G is assumed to be 2-connected. If G is planar, then it is clearly projective planar.

Algorithm 5.1 ([30]) *Projective Plane Embedding Algorithm.*

Input: *A 2-connected graph $G = (V, E)$ represented by adjacency lists, $|V| = n$, $|E| = m$*

Output: *Projective planar embedding of G or G is not projective planar*

if $m > 3n - 3$ **then**
  return(not projective planar)
if G is planar **then**
  return(planar embedding of G)
else *G is non-planar*
  Find a subgraph K in G that is a subdivision of K_5 or $K_{3,3}$.
for each labelled projective planar embedding K' of K **do**

Find all the bridges of K' and

determine which faces they can be embedded in.

if a bridge b cannot be embedded in any face of K' **then**

return(not projective planar)

Compute the conflicts between pairs of bridges in the faces

for each arrangement of 3-face bridges **do**

if there is an assignment of bridges to the faces of K' **then**

return(projective planar embedding of G)

return(not projective planar)

5.1.5 Analysis and Complexity of the Algorithm

If graph G has more than $3n - 3$ edges, it cannot be projective planar, and the algorithm returns. For the other steps of Algorithm 5.1, G has at most $3n - 3$ edges, i.e. $O(n)$ edges.

If graph G is planar, it is also projective planar. We can easily transform a planar rotation system of 2-connected graph G into a rotation system for a 2-cell embedding of the graph on the projective plane as described in Section 4.3. If G is not planar, then we can find a subgraph TK in G in $O(n)$ time by modifying a planarity algorithm as in [40].

The external loop is executed at most six times when $TK = TK_{3,3}$ and at

most 27 times when $TK = TK_5$ by Theorem 5.1. In either case it is executed a constant number of times.

The bridges can be found in $O(n)$ time by using a breadth-first or depth-first search. For a bridge B, it is possible to use a planarity testing algorithm to determine the faces in which it can be embedded. The planarity testing of bridges in a face can be done in the total computational time of $O(n)$. An embedding of $TK_{3,3}$ has four faces and an embedding of TK_5 has six faces. This gives $O(n)$ computational time for the bridge finding and testing embeddability in the faces.

An approach to find pairs of conflicting bridges is given in Section 5.1.3. Its time complexity is $O(n^2)$. The internal loop runs a constant number of times as discussed in Section 5.1.2. Every run consists of solving a 2-SAT problem instance in $O(n^2)$ time.

For the output, the projective planar embedding can be described by indicating planar embeddings of the bridges in the faces. To actually embed bridges inside of face F, we can connect a new vertex w to each vertex on the face boundary, add the bridges and invoke a planarity testing algorithm. Without loss of generality, we can assume that vertex w is embedded outside of the face boundary. Then bridges embedded outside of the face boundary can have the attachment points just in two adjacent vertices on the face boundary. We can easily flip these bridges inside of F to obtain a planar embedding of F with

all the bridges embedded inside. Therefore it takes $O(n)$ time to obtain the projective planar embedding of G given an assignment of bridges to the faces.

The conflicting bridges calculations for 2-SAT provides the worst case running time for the whole algorithm. Therefore the running time complexity of the algorithm is $O(n^2)$.

5.2 Practical Toroidality Testing

The toroidality testing algorithm of [31] is an exhaustive search method with a number of preliminary checks for sufficient properties of non-toroidal graphs. It is based on the properties of cycles in a graph G embedded in the torus. This is the only known implemented toroidality testing algorithm. This section provides the main ideas of the algorithm.

5.2.1 Flat Cycles and Non-Toroidal Graph Constraints

Here we describe some properties of non-toroidal graphs for a fast preliminary detection of such graphs. This simplifies the main exponential time algorithm which is an exhaustive search.

Definition 5.5 A cycle C of toroidal graph G is *flat* with respect to the surface if it is contractible in every torus embedding of G.

Let C be a cycle in graph G. The following theorem provides a characterization of flat cycles in a toroidal graph.

Theorem 5.2 ([31]) *A cycle C is flat in the toroidal graph G if deleting the vertices of C from G gives a non-planar graph $G\backslash C$. Furthermore, if the flat cycle C is chordless and $G\backslash C$ is connected, then C is a facial boundary in any 2-cell embedding of G on the torus.*

Proof. If the cycle C is not flat, then there exists a toroidal embedding of G such that C is essential. Such an embedding of G has $G\backslash C$ embedded on the cylinder and therefore planar. Otherwise, by Theorem 2.3, a non-planar $G\backslash C$ contains a subgraph of TK_5 or $TK_{3,3}$. All the faces in any embedding of TK_5 or $TK_{3,3}$ on the torus (see Figure 9.1 and Figure 10.1) are 2-cell. The cycle C must be added into the interior of a face in an embedding of TK_5 or $TK_{3,3}$. Therefore C is contractible in any embedding of G on the torus.

If C is chordless and $G\backslash C$ is connected, then C must be embedded in the interior of a face in an embedding of $G\backslash C$. In this case nothing can be embedded in the interior of C. ■

According to Theorem 2.2, Euler's formula for a 2-cell embedding of a connected graph G with n vertices, m edges and f faces on the torus is

$$n - m + f = 0.$$

Euler's formula gives the following two constraints for embedding on the torus.

Property 5.1 *If $m > 3n$, then G is non-toroidal.*

Proof. This property is a corollary of Euler's formula of Theorem 2.2. Given a 2-cell toroidal embedding, we can add edges while preserving toroidality by triangulating 2-cell faces. A triangulated 2-cell embedding has $m = 3n$ edges. The inequality follows. ∎

Property 5.2 *If $f > m - n$ for any embedding of G, then G is non-toroidal.*

Property 5.2 is also a corollary of Euler's formula for the torus. We can determine some 2-cell faces by chordless flat cycles using Theorem 5.2. The number of 2-cell faces in Theorem 5.2 gives a lower bound for the number of faces f. Also, given a rotation system for an embedding of G on a surface, we can easily count its number of faces f.

The 2-cell faces of Theorem 5.2 provide some other constraints for any embedding of graph G on the torus. Precisely, if vertices a and b are adjacent to vertex v on a 2-cell face boundary, then we write $(a, b)_v$ to indicate that a

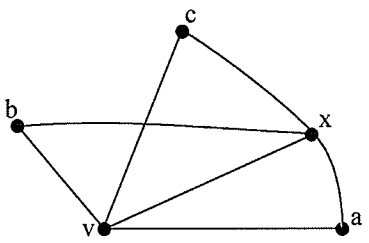

Figure 5.3: Edge vx overused by toroidal constraints

and b must appear consecutively in a cyclic adjacency list of v for a toroidal rotation system.

Property 5.3 *Any edge $\{x, v\} \in G$ occurs on the boundary of at most two faces of an embedding of G on a surface.*

Clearly, if an edge $\{x, v\} \in G$ must occur on the boundary of three or more faces for an embedding on the torus, the graph is non-toroidal. Property 5.3 is used by the algorithm in [31] in the following way. If we find an edge $\{x, v\} \in G$ that is shared by at least three 2-cell faces given by chordless flat faces of Theorem 5.2, then we have a set of constraints $\{(x, a)_v, (x, b)_v, (x, c)_v\}$ which is incompatible with any torus embedding (see Figure 5.3).

Property 5.4 ([31]) *If there exists a cyclic set of constraints $\{(a_1, a_2)_v, (a_2, a_3)_v, \ldots, (a_k, a_1)_v\}$ of size $k < deg(v)$ for an embedding of vertex $v \in G$ on the torus, then G is non-toroidal.*

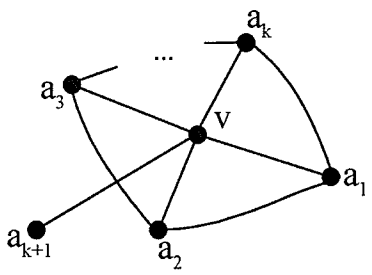

Figure 5.4: Embedding for a cyclic set of toroidal constraints

Proof. A rotation system contains all $deg(v)$ vertices adjacent to v in cyclic order. Therefore the cyclic set of constraints $\{(a_1, a_2)_v, (a_2, a_3)_v, \ldots, (a_k, a_1)_v\}$, where $k < deg(v)$, $v \in G$, is not compatible with any toroidal rotation system of G (see Figure 5.4). ■

Notice that for a toroidal embedding of G, if vertex v has $deg(v)-1$ compatible constraints or $deg(v)-2$ consecutive compatible constraints, we can complete them to the cyclic set of constraints of size $deg(v)$.

Property 5.1 can be checked directly from the graph input information. To detect a non-toroidal graph faster, we can take a collection of short cycles and use Theorem 5.2 to find 2-cell faces among them. The 2-cell faces provide sets of constraints. We can use the information to check Properties 5.2 - 5.4 for non-toroidal graphs.

In some cases, the determination of 2-cell faces using Theorem 5.2 results in a complete set of constraints. Then it is possible to unify the constraints to

obtain a toroidal embedding of the graph or to conclude that none exists in $O(n)$ time.

5.2.2 Essential Cycles and Toroidality Testing

The main strategy of the algorithm is to reduce the toroidality problem to planarity testing. The first step is to find an essential cycle C in the graph G for its embedding on the torus. We assume the graph is toroidal and the essential cycle C cuts the torus into a cylindrical surface as described in Chaper 4.

Since we do not know precisely which cycles of G are essential, we need to try several candidates. The number of candidates for an essential cycle is relatively small, either $O(n)$ by Theorem 4.2, or $O(1)$ by Corollary 4.2. Since by Theorem 2.3, a non-planar toroidal graph G contains a subdivision TK_5 or $TK_{3,3}$, we can use Corollary 4.2 as well. The theorem guarantees that at least one of the cycles must work.

Now we assume G could be embedded on the torus with the cycle C embedded as an essential cycle. According to Chapter 4, C cuts the torus into a cylinder. This corresponds to cutting G on the torus along the cycle C, duplicating cycle C as C_1 and C_2, and marking the interior of two copies of C as forbidden regions for any embedding of G (see Figure 5.5). This gives a

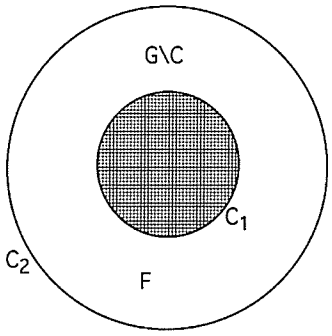

Figure 5.5: Cylinder embedding for G cut along cycle C

cylinder embedding. Now suppose there is an embedding of G in the plane such that attaching the edges incident to cycle C either to copy C_1 or C_2 gives a cylinder embedding of Figure 5.5. Then gluing the two copies C_1 and C_2 back to C gives a torus embedding of the initial graph G (see Figure 5.5).

There are two problems with the cylinder embedding. First of all, we do not know which copy C_1 or C_2 of C an edge of G should be incident to. To cover all possible cases, the implemented algorithm checks all bipartitions for the edge endpoints on cycle C. This makes the algorithm exponential in time.

The second problem is the initial cylinder face F between two copies C_1 and C_2 of C. Face F is not equivalent to an open disk, therefore it is not possible to use planarity testing directly to add $G\backslash C$ into F. However if we can find two edge disjoint paths p_1 and p_2 having one end on C_1 and the other end on C_2, the paths would divide face F into two faces F_1 and F_2 both equivalent to

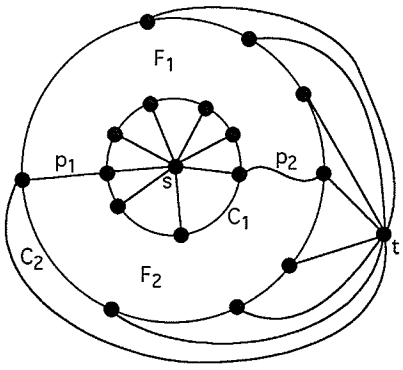

Figure 5.6: Two edge disjoint paths to cut the cylinder face

an open disk without repeating vertices on the face boundary (see Figure 5.6).

For faces F_1 and F_2 it is possible to use planarity testing to embed $G \backslash (C \cup p_1 \cup p_2)$. To find two edge disjoint paths p_1 and p_2 between C_1 and C_2, two new vertices s and t are added so that vertex s is incident to all vertices of C_1 and t is incident to all vertices of C_2. Now it is possible to use a standard flow algorithm to find two edge disjoint (s,t)-paths. This provides paths p_1 and p_2 necessary for planarity testing. If there are no two edge disjoint paths p_1 and p_2 between C_1 and C_2, then, by Menger's theorem [4], there is a cut vertex w. It is possible to split the graph into two pieces at the cut vertex w, then to decide on a planar embedding of each piece separately and to glue the pieces together to get a cylinder embedding of G.

Since we need to check all the bipartitions of edges incident on the vertices of C, it is reasonable to select cycles C with few incident edges. For this pur-

pose, cycles provided by Theorem 4.2 are better than cycles of Corollary 4.2. To choose a cycle basis of Theorem 4.2 that minimizes the exponential time component of the algorithm, Myrvold and Neufeld [31] use Horton's algorithm of [20].

Chapter 6

Graphs Containing K_5-Subdivisions

Let G be a non-planar graph. By Kuratowski's Theorem [25], G contains a subdivision of K_5 or $K_{3,3}$ as a subgraph. We denote by TK_5 a subdivision of K_5 and by $TK_{3,3}$ a subdivision of $K_{3,3}$ in G.

Definition 6.1 The vertices of degree 4 or 3 are *corners* of TK_5 or $TK_{3,3}$ respectively and the vertices of degree 2 are *inner vertices* of TK_5 or $TK_{3,3}$.

The corners of TK_5 or $TK_{3,3}$ have been called *main vertices* in [28], [21] and [30].

Definition 6.2 A path of TK_5 or $TK_{3,3}$ whose endpoints are two distinct corners and all other vertices are inner vertices of TK_5 or $TK_{3,3}$ is called a *side* of TK_5 or $TK_{3,3}$.

Notice that two sides of TK_5 or $TK_{3,3}$ can have at most one common corner and no common inner vertices.

Definition 6.3 A side having a common corner with another side of TK_5 or $TK_{3,3}$ is called *adjacent* to that side. Two sides having no common corner are called *non-adjacent*.

6.1 Short Cuts and 3-Corner Vertices

Suppose G contains a subdivision TK_5 as a subgraph. This section describes how the subdivision TK_5 can be transformed into a subdivision $TK_{3,3}$ in G. Notice that a pair of corners of TK_5 determines a unique side of TK_5, an inner vertex is on a uniquely determined side and each corner is on exactly four distinct sides.

Definition 6.4 A path P in G whose one endpoint u is an inner vertex of TK_5, the other endpoint is not on the side of u, and all other vertices and edges are in $G \backslash TK_5$ is called a *short cut* of the K_5-subdivision.

Definition 6.5 A vertex $u \in G \backslash TK_5$ is called a 3-*corner vertex* with respect to TK_5 if $G \backslash TK_5$ contains internally disjoint paths from u to at least three corners of the K_5-subdivision (see Figure 6.4).

We begin by proving some basic structural results for graphs containing a TK_5. Similar structural results have been proved by M. Fellows and P. Kaschube in [10]. Notice that the proof of Theorem 1 in [10] is missing the case indicated by Figure 6.1 of Proposition 6.1.

Proposition 6.1 ([10]) *A non-planar graph G with a K_5-subdivision TK_5 for which there is either a short cut or a 3-corner vertex contains a $K_{3,3}$-subdivision.*

Proof. To prove the proposition, we exhibit a $K_{3,3}$-subdivision in G. In the following diagrams the bipartition of $K_{3,3}$ is indicated by black and white vertices. Vertices which are not part of $K_{3,3}$ are shaded grey.

The following cases are possible.

Case 1. Both endpoints of a short cut P are inner vertices of TK_5 and the corresponding two sides are non-adjacent. Figure 6.1 shows a $K_{3,3}$-subdivision in G.

Case 2. Both endpoints of a short cut P are inner vertices of TK_5 and the corresponding two sides are adjacent. Figure 6.2 shows a $K_{3,3}$-subdivision in

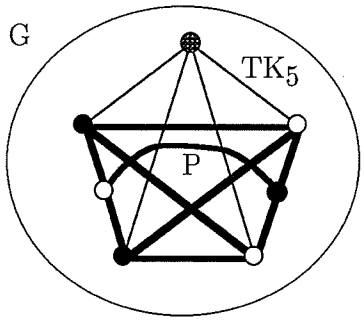

Figure 6.1: $K_{3,3}$ created by short cut P

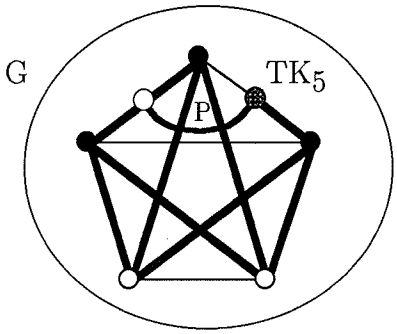

Figure 6.2: $K_{3,3}$ created by short cut P

the graph G.

Case 3. One of the endpoints of a short cut P is a corner of TK_5. Figure 6.3 shows a $K_{3,3}$-subdivision in G.

Now suppose there is a 3-corner vertex $u \in G \backslash TK_5$. Then Figure 6.4. shows a $K_{3,3}$-subdivision in G.

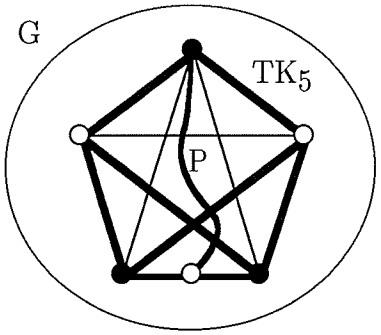

Figure 6.3: $K_{3,3}$ created by short cut P

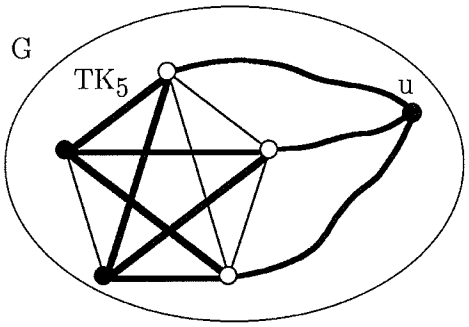

Figure 6.4: $K_{3,3}$ created by 3-corner vertex u

Thus any short cut or 3-corner vertex of TK_5 in G gives a $K_{3,3}$-subdivision. ■

6.2 Side Components

Let G be a graph having no 3-corner vertex and no short cut of TK_5. Denote by K the set of corners of TK_5. Consider the set of connected components of $G\backslash K$. Let C be any connected component of $G\backslash K$.

Proposition 6.2 ([10]) *For a graph G with TK_5 and no short cut or 3-corner vertex of TK_5, a connected component C of $G\backslash K$ contains inner vertices of at most one side of TK_5. Moreover vertices of C are adjacent in G to exactly two corners of TK_5.*

Proof. Suppose a connected component C contains inner vertices of two different sides of TK_5. Then clearly C contains a short cut of TK_5 in G, a contradiction.

Suppose C has vertices which are adjacent to at least three different corners of TK_5 in G. Then it is not difficult to see that there is either a short cut or a 3-corner vertex of TK_5 in G, a contradiction. Therefore, vertices of C are adjacent to at most two corners of TK_5 in G. Since G is 2-connected, there are exactly two corners of TK_5 adjacent to vertices of C in G. ■

Definition 6.6 Given a graph G without a short cut or a 3-corner vertex of TK_5 we define a *side component* of TK_5 as a subgraph in G induced by a pair of corners a and b of TK_5 and all connected components of $G\backslash K$ which have vertices adjacent to the two corners a and b in G.

Corollary 6.1 *Two side components of TK_5 in G have at most one vertex in common. The common vertex is the corner of intersection of two corresponding sides of TK_5.*

Proof. Any pair of corners of TK_5 defines a side. Since G is 2-connected, by Proposition 6.2, we can associate every connected component of $G\backslash K$ with a unique side of TK_5. This gives a partition of vertices of $G\backslash K$ into side components of TK_5. ■

Notice that side components, however, can contain a $K_{3,3}$-subdivision. Thus, given a graph G with a K_5-subdivision TK_5, either we can find a short cut or a 3-corner vertex of TK_5 in the graph, or else we can partition the vertices and edges of $G\backslash TK_5$ into equivalence classes according to the corresponding side components of TK_5 in G.

6.3 Augmented Side Components

Every side component H of TK_5 contains exactly two corners a and b corresponding to a side of TK_5. If edge ab between the corners is not in H, we can add it to H to obtain $H + ab$. Otherwise $H + ab = H$.

Definition 6.7 Given two corners a and b of a side component H, edge ab is called the *corner edge* of $H + ab$ and $H + ab$ is called an *augmented side component* of TK_5.

We use the following general lemma for side components of a K_5-subdivision in the embedding algorithms. By the lemma, a corner edge can be added into every side component H to test easily if there exists an embedding of H with both corners on the outer face and to find such an embedding. The lemma presents a well known fact for planar embeddings. The use of the lemma and the augmented side components will be explained in Section 7.1 and Chapter 9.

Lemma 6.1 *There is a planar embedding of a graph G with two vertices u and v on the outer face if and only if there exists a planar embedding of the graph $G + uv$.*

Proof. It can be seen by drawing any planar embedding on the sphere that any face of a planar embedding can be considered as an outer face. Now if

there exists an embedding of graph G on the sphere with both vertices u and v on the same face, then we can just add the edge between them into the face. Otherwise for any embedding of G on the sphere the edge cannot be added into the planar embedding. Hence $G + uv$ is not planar. ■

Chapter 7

Embedding Graphs on the Projective Plane

7.1 K_5-Subdivisions and Planarity

The algorithm in this section applies standard planarity techniques to find a Kuratowski subgraph in a graph G. If the found subgraph is TK_5, then either Proposition 6.1 is used to find a $TK_{3,3}$ subgraph, or Proposition 6.2 applies to reduce the projective planarity determination to the planarity checks. This provides a linear time practical algorithm to check if a non-planar graph G is projective planar or if it contains a $K_{3,3}$-subdivision.

7.1.1 Characterization for Projective Planarity Checking

Let G be a 2-connected non-planar graph with a K_5-subdivision TK_5. We begin with a characterization of projective planarity for graphs with a K_5-subdivision.

Theorem 7.1 *A graph G with a K_5-subdivision TK_5 and no short cut or 3-corner vertex of TK_5 is projective planar if and only if all the augmented side components of TK_5 are planar graphs.*

Proof. By Corollary 6.1, all the vertices and edges of $G \backslash TK_5$ are partitioned into side components. The sufficient and necessary conditions of the theorem can be proved as follows.

First we show that the sufficient conditions hold. Take any embedding of TK_5 on the projective plane (see Figure 7.1). For each side of TK_5, make a planar embedding of its side component with both corners on the outer face. By Lemma 6.1, there exists such an embedding of a side component if and only if the augmented side component is a planar graph. By Corollary 6.1, we can embed every side component independently.

Now we prove the necessary conditions of the theorem. Figure 7.1 shows the two possible non-isomorphic embeddings of TK_5 on the projective plane (see

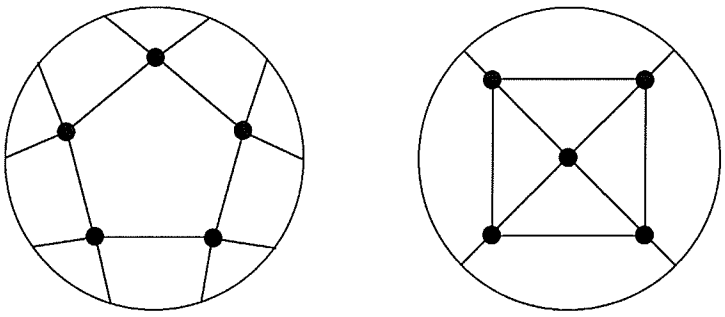

Figure 7.1: K_5 on the projective plane

[28] and [30] for details).

The sides of TK_5 must create one of these embeddings. Each embedding divides the projective plane into faces. Each vertex of TK_5 appears at most once on the boundary of any face, and every side of TK_5 is incident to exactly two faces. Call these faces F_1 and F_2, and let K be the set of corners of TK_5. For some sides, it is possible that the two corners a and b also appear on the boundary of a third face, as non-consecutive vertices. But since G has no short cut or 3-corner vertex of TK_5, every connected component C of $G\backslash K$, adjacent to a and b and embedded in a third face can also be embedded in F_1 or F_2. This shows that it is always possible to embed every side component of TK_5 in an open disk contained in $F_1 \cup F_2$, i.e. every augmented side component must be planar. ■

7.1.2 Graphs with a K_5-Subdivision

Let G be a 2-connected non-planar graph with a K_5-subdivision TK_5. Theorem 7.1 provides the basis for a linear time practical algorithm to check if the graph is projective planar or if it contains a $K_{3,3}$-subdivision.

Algorithm 7.1 *Embedding Graphs with a K_5-Subdivision on the Projective Plane.*

Input: *A 2-connected graph G*

Output: *Either a projective planar rotation system of G, or a $K_{3,3}$-subdivision in G, or an indication that G is not projective planar*

(1) Use a linear time planarity checking algorithm (eg. [19], [33], [23], [5], [39] and [40]) to determine if G is planar. If G is planar then return its planar rotation system. If G is not planar and the planarity check returns a $K_{3,3}$-subdivision in G then return the $K_{3,3}$-subdivision in G.

(2) If G is not planar and the planarity check returns a K_5-subdivision TK_5 in G, then do a depth-first or breadth-first search to look for either a short cut or a 3-corner vertex of TK_5 in G. If a short cut or a 3-corner vertex is found, then return a $K_{3,3}$-subdivision in G as per Proposition 6.1. If there are no short cut or 3-corner vertices, the depth-first or breadth-first search returns the side components of TK_5.

(3) For each side component H of TK_5 in G, if it is necessary, augment H by adding the corner edge ab to have $H + ab$, and check if $H + ab$ is planar. If all the augmented side components are planar, then return a projective planar rotation system of G. If there is a non-planar augmented side component of TK_5, then return G is not projective planar.

Every step in this algorithm has linear time complexity. Therefore the entire algorithm is also linear.

7.2 A Spanning $K_{3,3}$-Subdivision

In this section we describe the possible embeddings of a $K_{3,3}$-subdivision on the projective plane and the possible ways to complete an embedding to a projective planar graph G. First we cover the case where a $K_{3,3}$-subdivision $TK_{3,3}$ is a spanning subgraph in G. In Section 8.3 we describe how to generalize it to an arbitrary $K_{3,3}$-subdivision in G. A generalization for a non-spanning $K_{3,3}$-subdivision in G can be done by analogy with the recursion in the Hopcroft-Tarjan algorithm of Chapter 3. This section describes the structure of the projective planar graphs with respect to the spanning $K_{3,3}$-subdivision.

Let G be a non-planar graph with a spanning $K_{3,3}$-subdivision $TK_{3,3}$. We assume that $K_{3,3}$ has a bipartition of its vertices labelled as $\{a_1, a_2, a_3\}$ and $\{b_1, b_2, b_3\}$ (see Figure 7.2).

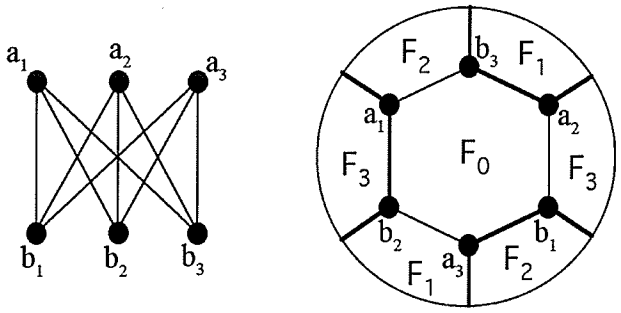

Figure 7.2: $K_{3,3}$ and its embedding on the projective plane

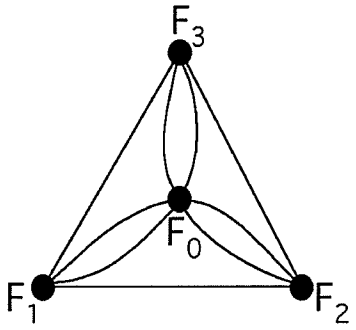

Figure 7.3: The dual graph of the projective planar embedding of $K_{3,3}$

Figure 7.2 shows the unique embedding of $K_{3,3}$ on the projective plane. The dual graph for the embedding of $K_{3,3}$ of Figure 7.2 is shown in Figure 7.3. Notice that Figure 7.3 depicts a planar drawing of the dual graph, not a projective planar embedding. Therefore it does not show mutual positions of the faces of Figure 7.2 on the projective plane.

The embedding of Figure 7.2 is a 2-cell embedding and, according to Section

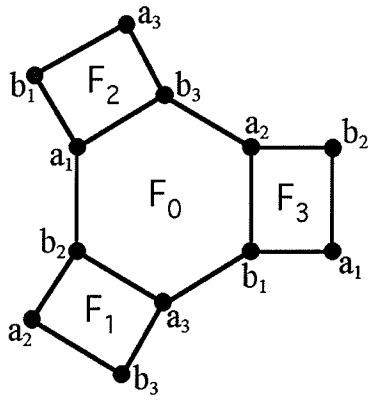

Figure 7.4: Unfolded faces of an embedding of $TK_{3,3}$

4.1, there is a cycle of $K_{3,3}$ that is essential for the embedding. An essential cycle crosses the projective plane boundary an odd number of times. $K_{3,3}$ has six distinct hamiltonian cycles. Each hamiltonian cycle C of $K_{3,3}$ corresponds to a perfect matching $K_{3,3}\backslash EC$. The embedding of Figure 7.2 has two essential hamiltonian cycles and four contractible hamiltonian cycles. Therefore in a projective planar embedding of a graph G with a spanning $K_{3,3}$-subdivision $TK_{3,3}$, the $TK_{3,3}$ always has an essential cycle which corresponds to an essential hamiltonian cycle of the embedding of $K_{3,3}$.

Figure 7.4 shows the embedding of Figure 7.2 cut along the essential hamiltonian cycle $C = (a_1b_1a_3b_3a_2b_2)$ of $K_{3,3}$ of Figure 7.2. We say that the faces of $TK_{3,3}$ have been *unfolded.*

An embedding of $TK_{3,3}$ on the projective plane has one face bounded by 6

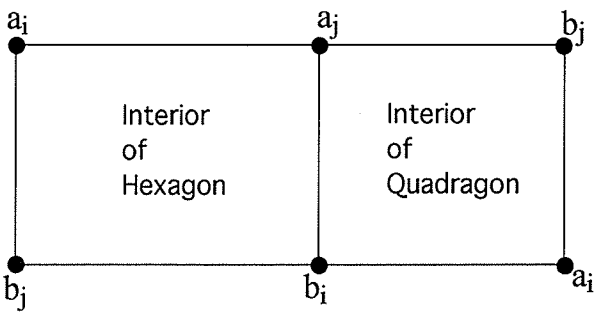

Figure 7.5: Möbius band cut along a side

sides and three faces bounded by 4 sides of $TK_{3,3}$. We call the 6-sided face the *hexagon*, and the 4-sided faces the *quadragons* of a $TK_{3,3}$-embedding. In the diagram of Figure 7.4, the hexagon is denoted by F_0 and each quadragon is denoted by F_i, $i = 1, 2, 3$, where i is the missing subscript of the corners of the quadragon boundary. We call the sides a_ib_i, $i = 1, 2, 3$, *quadragon sides*, and the sides a_ib_j, $i, j = 1, 2, 3$, $i \neq j$, *hexagon sides*.

Definition 7.1 Given an embedding of $TK_{3,3}$ on the projective plane with the labelling of Figure 7.4, a *Möbius band* consists of two opposite hexagon sides a_ib_j and b_ia_j, $i, j = 1, 2, 3$, $i \neq j$, plus the interior of the hexagon and the interior of the quadragon (see Figure 7.5).

Figure 7.5 shows a Möbius band cut along the side a_ib_j. Each Möbius band has two parts corresponding to the two faces. One part contains the two sides appearing on its boundary in cyclic order as a_ib_j and b_ia_j, and the other part

in cyclic order as a_ib_j and a_jb_i, $i, j = 1, 2, 3$, $i \neq j$. There are three different Möbius bands, corresponding to the 3 pairs of opposite sides of the hexagon. All three Möbius bands share the hexagon interior (face F_0 in Figure 7.4), but have different quadragons.

7.2.1 The Labelled Embeddings of $TK_{3,3}$

There are six different ways to label the corners of an embedding of $TK_{3,3}$ on the projective plane (see Figure 7.6). The labellings of Figure 7.6 can be obtained from the labelling of Figure 7.2 by successively applying the permutation $(a_1)(a_2a_3)(b_1b_2b_3)$. This is a convenient way to obtain all the labellings in a computer program, starting from the initial labelling of Figure 7.2.

Since $TK_{3,3}$ is supposed to be spanning, it is necessary to determine if it is possible to add the remaining edges of G, without crossing, into the 4 faces for at least one of the 6 labelled embeddings. The labelled embeddings have distinct closed walks on the face boundaries. They correspond to 6 different hamiltonian cycles of $K_{3,3}$ appearing as the hexagons. The example of Figure 7.8 shows that all the six labelled embeddings must be considered as a possible initial embedding of $TK_{3,3}$ to complete it to an embedding of a graph G. The example of Figure 7.8 will be considered in detail in Subsection 7.2.2.

Without loss of generality, assume the embedding of $TK_{3,3}$ is labelled as the

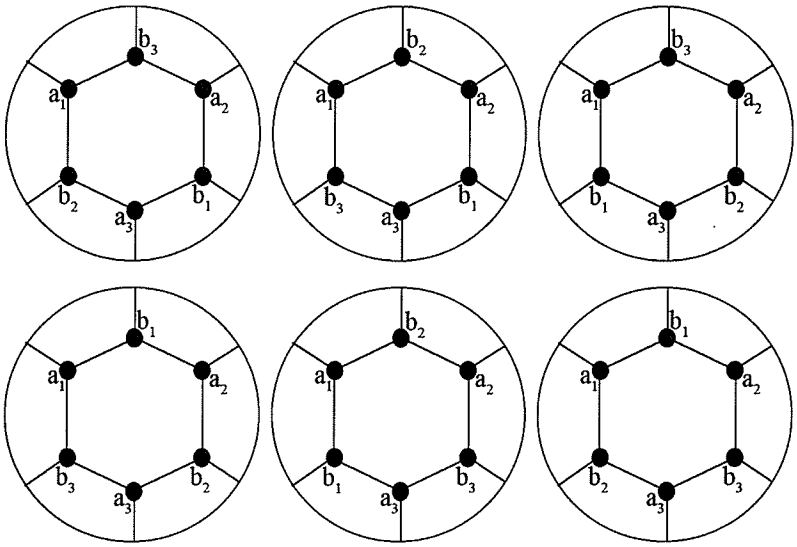

Figure 7.6: The six labelled embeddings of $TK_{3,3}$

first one in Figure 7.6. We will consider this labelled embedding in more detail. The results and discussion are true for any embedding of $TK_{3,3}$.

7.2.2 Chords and Faces

This subsection describes different types of edges of $G \backslash TK_{3,3}$ and possible ways to add them to an embedding of $TK_{3,3}$ on the projective plane. Later subsections will provide more detail for each particular type and for the interplay between different types.

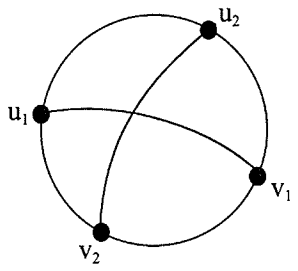

Figure 7.7: Two chords crossing in a face

Definition 7.2 An edge of $G \backslash TK_{3,3}$ is called a *chord.*

Definition 7.3 Two chords u_1v_1 and u_2v_2 are said to *cross in face* F iff they are disjoint and their endpoints alternate on the boundary of F in a cyclic order as u_1, u_2, v_1, v_2, clockwise or counter-clockwise (see Figure 7.7).

Clearly, two adjacent chords with 3 different endpoints on a face boundary can always be drawn in the face without crossing. Similarly, two disjoint chords with their endpoints on a face boundary in a non-alternating cyclic order can always be drawn in the face interior without crossing.

We will consider the unfolded faces of Figure 7.4 corresponding to the embedding of Figure 7.2 in more detail to develop an algorithm which efficiently decides if we can complete an embedding without chords crossing.

Definition 7.4 A chord is called a *k-face chord* if it admits an embedding in exactly k different faces of the embedding of $TK_{3,3}$ on the projective plane (see Figure 7.2 and Figure 7.4).

Proposition 7.1 *Referring to the labelling of Figure 7.4, a chord uv is a*

(i) 3*-face chord if and only if* $uv = a_ib_i$, $i = 1, 2, 3$;

(ii) 2*-face chord if and only if both endpoints u and v are on the same side except* $uv = a_ib_i$, $i = 1, 2, 3$, *or one endpoint of uv is on the side with corners* a_i *and* b_j *and the other is on the side with corners* b_i *and* a_j, $i, j = 1, 2, 3$, $j \neq i$;

(iii) 0*-face chord if and only if one endpoint is in the interior of the side with endpoints* a_i *and* b_i, $i = 1, 2, 3$, *and the other is in the interior of a side* a_jb_k, $j, k = 1, 2, 3$, *where* $j \neq i$, $k \neq i$ *and* $j \neq k$.

Otherwise uv is a 1*-face chord.*

Proof. Consider all the possible cases for adding a chord in the diagram of Figure 7.4 with respect to the side labelling. Any chord incident to a corner has "access" to all sides of the hexagon and quadragons. Any chord having both endpoints on the same side can be placed in two faces separated by the side. This gives 2-face chords. However chords a_ib_i, $i = 1, 2, 3$, having both

endpoints in the side corners can be placed in the hexagon as well. This gives 3-face chords.

Any chord with an endpoint on a hexagon side and the other endpoint on an opposite hexagon side can be placed in a quadragon as well. This gives 2-face chords.

Any chord having an endpoint in the interior of a quadragon side a_ib_i, $i = 1, 2, 3$, does not have "access" to the interior of a hexagon side not incident to the corners a_i and b_i. This gives 0-face chords.

In all the remaining cases a chord can be placed in a unique face. ■

Proposition 7.1 gives us a characterization of the chords. We can determine if uv is a k-face chord, $k = 0, 1, 2, 3$, by checking its endpoints with respect to the sides and their subscript labelling. This can be done efficiently in a computer program. If a 0-face chord is detected for an embedding of $TK_{3,3}$, then the embedding is not projective planar.

Before we analyze the chords with respect to a labelled embedding of $TK_{3,3}$, we show an example that all six labelled embeddings of Figure 7.6 are important and must be considered to obtain an embedding of a graph G. Figure 7.8 illustrates a graph $G = TK_{3,3} \cup \{e_1, e_2, e_3\}$ whose embedding on the projective plane can be obtained from just one labelled embedding of Figure 7.6. The graph G is obtained from $TK_{3,3}$ by adding three chords $e_1 = xy$, $e_2 = wz$

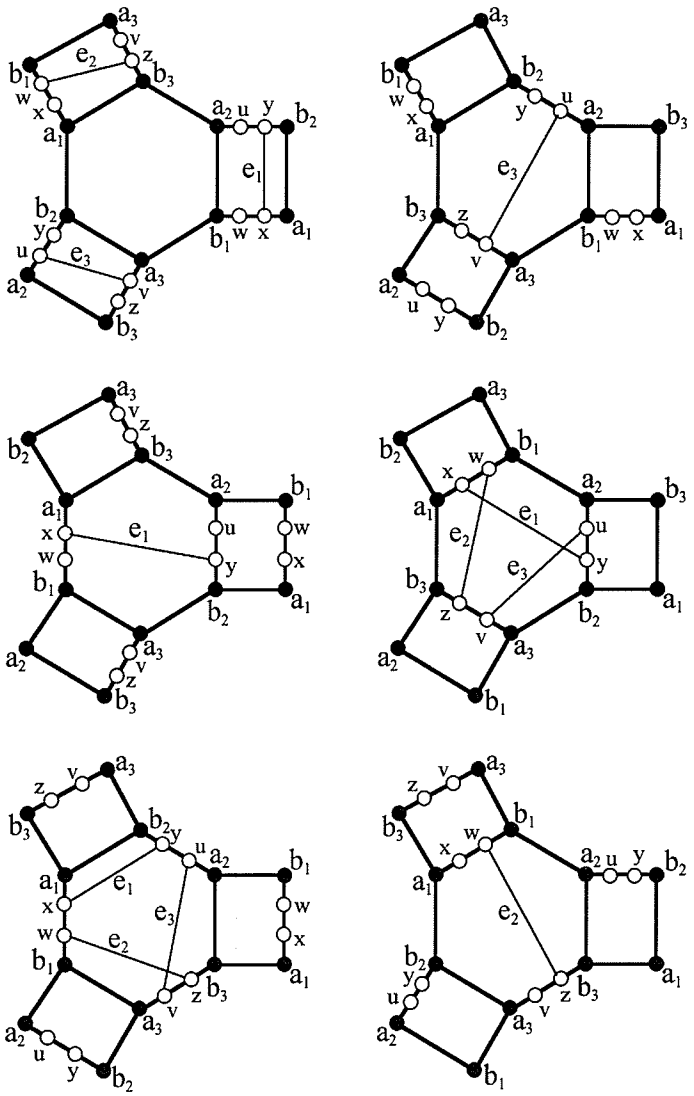

Figure 7.8: The unique embedding of $G = TK_{3,3} \cup \{e_1, e_2, e_3\}$

and $e_3 = uv$ in the quadragons of the first diagram of Figure 7.8. The other labellings of $TK_{3,3}$ either have one of e_1, e_2, e_3 as a 0-face chord, or else have a crossing of 1-face chords e_1, e_2, e_3. A similar example can be constructed for every labelled embedding of $TK_{3,3}$.

7.2.3 1-Face Chords and Forced Chords

Figure 7.9 shows the pattern of 1-face chords drawn in the faces of Figure 7.4. Clearly, a 1-face chord must be placed in a unique face. If any two 1-face chords cross in some face, then the current embedding of $TK_{3,3}$ can not be extended to G. We can check this by examining the endpoints of a 1-face chord on the face boundary with respect to the other 1-face chords.

Now suppose some of the six labelled embeddings of $TK_{3,3}$ have no two 1-face chords crossing. Then it is necessary to decide efficiently if there is a way to add 2-face and 3-face chords into such an embedding.

Uniquely Embeddable Chords Generated by 1-Face Chords

Let e be a 2-face or 3-face chord. Such a chord e can be placed in several faces of the $TK_{3,3}$-embedding. However a 1-face chord or another uniquely embeddable chord may cross some possible embeddings of e, forcing e to have

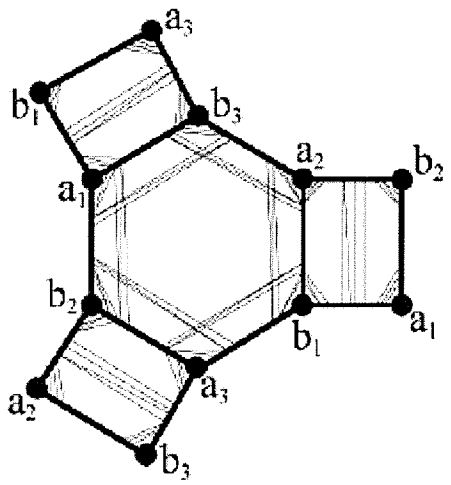

Figure 7.9: The pattern of 1-face chords

at most one permitted embedding remaining.

Definition 7.5 Given a labelled embedding of $TK_{3,3}$, a chord e is called *forced* if it is either

(i) a 1-face chord, or

(ii) e has just one embedding which does not cross a 1-face chord or another previously forced chord.

Forced chords are generated by 1-face chords and behave exactly like 1-face chords. They must be embedded in the unique face which results in no conflicts

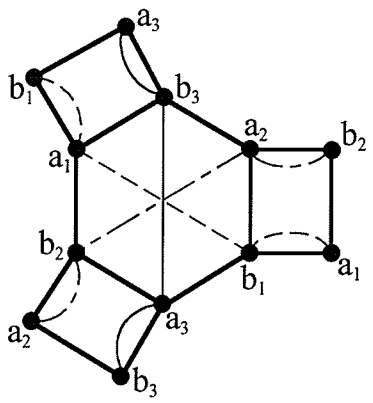

Figure 7.10: The pattern of 3-face chords

with other forced chords.

7.2.4 3-Face Chords

According to Proposition 7.1, an embedding of $TK_{3,3}$ admits at most three 3-face chords corresponding to three pairs of opposite corners of the hexagon or to three sides separating the quadragons. Figure 7.10 shows all possible embeddings for 3-face chords a_1b_1, a_2b_2 and a_3b_3.

Each 3-face chord can be embedded in one of two quadragons or in the hexagon.

Lemma 7.1 *A 3-face chord a_ib_i, $i = 1, 2, 3$ can be placed in a quadragon if and only if there is no 1-face chord having one endpoint which is an interior*

vertex of the side a_ib_i of the quadragon. At most one 3-face chord can be placed in the hexagon.

Proof. Referring to Figure 7.9, a 3-face chord a_ib_i, $i = 1, 2, 3$, may potentially cross only the 1-face chords in the quadragons. Therefore if there are no 1-face chords to cross the 3-face chord in a quadragon, we can embed it in the quadragon. Otherwise the 3-face chord is forced and must be embedded in the hexagon. As can be seen from Figure 7.10, all three 3-face chords cross each other in the hexagon. Therefore at most one of them can be placed in the hexagon. ■

As a result, we can decide on the embedding of a 3-face chord independently by trying embedding it into a quadragon if there is one which has no 1-face chords to cross the 3-face chord. If a 3-face chord cannot be embedded in a quadragon, it must be placed in the hexagon. A 3-face chord embedded in the hexagon causes a special case for an embedding which is discussed in Subsection 7.3.1 and is shown schematically in Figure 7.15.

7.2.5 2-Face Chords

In this section we consider 2-face chords that are not forced. These chords can be numerous and complicated to decide on their embedding. However we can classify them to show that it is possible to decide on their embedding

efficiently.

Definition 7.6 Two 2-face chords corresponding to the same two faces are said to be *in conflict* if and only if they cross when drawn in the same face. We define a *conflict graph* $H = (V(H), E(H))$, where $V(H)$ is a set of 2-face chords, i.e. $V(H) \subset E(G)$, and for $e_i, e_j \in V(H)$, there is an edge $\{e_i, e_j\} \in E(H)$ if and only if corresponding chords e_i and e_j are in conflict.

Clearly, for any embedding of graph G on the projective plane it is necessary for the conflict graph H to be bipartite. For a pair of 2-face chords in conflict, embedding one of them in one face forces the other one to be embedded in the other face to avoid a crossing.

Definition 7.7 A *bundle* is a set of 2-face chords corresponding to a connected component of the bipartite conflict graph H.

An embedding of one of the chords in a bundle determines the embedding of all other chords from the same bundle.

Quadragon 2-Face Chords

Definition 7.8 A 2-face chord that has both endpoints on a side a_ib_i, $i = 1, 2, 3$, separating two quadragons in Figure 7.4, is called a *quadragon* 2-

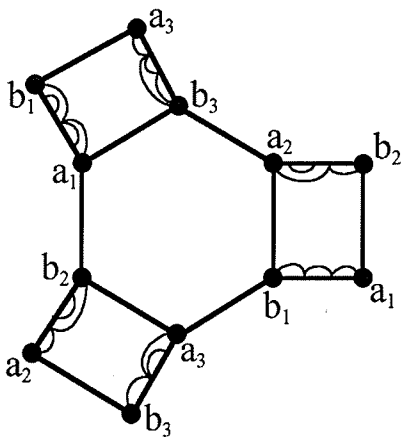

Figure 7.11: The pattern of quadragon 2-face chords

face chord (see Figure 7.11). A *group of quadragon 2-face chords* is a set of quadragon 2-face chords having both endpoints on the same side.

A quadragon 2-face chord can only be in conflict with quadragon 2-face chords from the same group. The other 2-face and 3-face chords can avoid a crossing with the quadragon 2-face chords by an appropriate drawing in the quadragon. There are exactly three sides separating three quadragons of the embedding. This provides three different groups of quadragon 2-face chords.

Lemma 7.2 *For any embedding of graph G, each group of quadragon 2-face chords admits a bipartition with one part embedded in one quadragon and the other part embedded in the other quadragon without crossing.*

Proof. Otherwise there would be two quadragon 2-face chords crossing in a quadragon. ■

A group of quadragon 2-face chords is completely determined by the chord endpoints on the same side a_ib_i, $i = 1, 2, 3$. We can decide separately on an embedding for each group of the quadragon 2-face chords by constructing a bipartition for its embedding.

Parallel and Perpendicular 2-Face Chords

In this section we consider 2-face chords having both endpoints on the hexagon boundary of Figure 7.4. Each of the chords can be placed either in the hexagon or in one of the quadragons. We can distinguish two types of the chords.

Definition 7.9 For an embedding of $TK_{3,3}$ on the projective plane, a 2-face chord having both endpoints on the same side of the hexagon is called *parallel* to the side (see Figure 7.12). A 2-face chord having its endpoints on two opposite sides of the hexagon is called *perpendicular* to the corresponding sides (see Figure 7.12). A perpendicular chord whose both endpoints are corners of $TK_{3,3}$ is called a *diagonal* (chord a_1a_3 in Figure 7.12).

For each labelled embedding of $TK_{3,3}$ on the projective plane, there are six different groups of parallel chords corresponding to the hexagon sides and

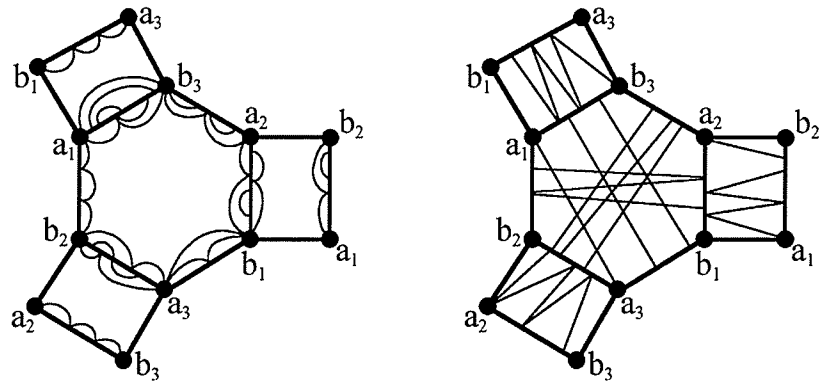

Figure 7.12: The pattern of parallel and perpendicular 2-face chords

three different groups of perpendicular chords corresponding to three pairs of opposite sides of the hexagon (see the embedding of Figure 7.2 and the unfolded embedding of Figure 7.4). There can be at most six diagonals, namely a_1a_2, b_1b_2, a_2a_3, b_2b_3, a_1a_3, b_1b_3 shown in Figure 7.13.

Parallel chords can be in conflict only with parallel chords or perpendicular chords having an endpoint on the same side. A perpendicular chord having both endpoints in interior vertices definitely crosses any perpendicular chord from a different group when embedded in the hexagon. However a perpendicular chord having an endpoint in a corner does not cross other parallel or perpendicular chords incident to the same corner. Also a perpendicular chord might interfere with the other chords of the same group and with parallel chords of two groups on opposite sides of the hexagon. Each parallel and perpendicular chord must be embedded in the hexagon or associated quadragon

of the $TK_{3,3}$ embedding (Figure 7.12).

7.3 The Möbius Band

In this section we consider the structure of chords in a Möbius band and the compatibility of two or three different Möbius bands. All three Möbius bands of Figure 7.4 share the hexagon interior. A parallel or a perpendicular chord can be embedded into any of two faces of its corresponding Möbius band.

Definition 7.10 A Möbius band is called *flat* if there exist an embedding of its parallel and perpendicular chords without crossing such that all the perpendicular chords are embedded in the quadragon (for example, see Figure 7.20 and Figure 7.21). Otherwise it is *non-flat* (for example, see Figures 7.19, 7.22). A Möbius band is called *diagonal compatible* if there exist an embedding of its parallel and perpendicular chords without crossing such that just one diagonal is embedded in the hexagon and all the other perpendicular chords are embedded in the quadragon (for example, see Figure 7.13). A Möbius band is called *corner compatible* if there exist an embedding of its parallel and perpendicular chords without crossing such that some perpendicular chords incident on one corner are embedded in the hexagon and all the other perpendicular chords are embedded in the quadragon (for example, see Figure 7.14). Otherwise it is *unmatchable* (for example, see Figures 7.19, 7.22, 7.23). A Möbius band is

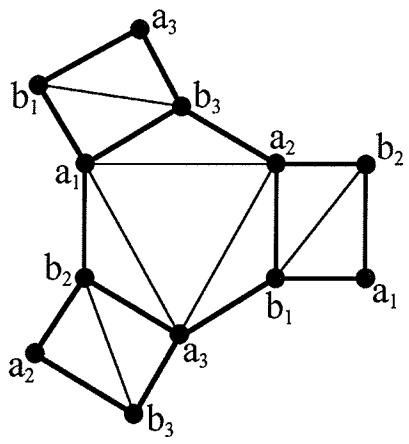

Figure 7.13: Three matched diagonal compatible Möbius bands

called *embeddable* if there exists an embedding of corresponding parallel and perpendicular chords without crossing.

Definition 7.11 Diagonal compatible Möbius bands are *matched* if they admit a simultaneous embedding without diagonals crossing. Two corner compatible Möbius bands are *matched* if they admit an embedding without perpendicular chords crossing in the hexagon. Otherwise they do not *match.*

Figure 7.13 shows three matched diagonal compatible Möbius bands. Notice that the Möbius bands of Figure 7.13 are matched and diagonal compatible for the other diagonals as well.

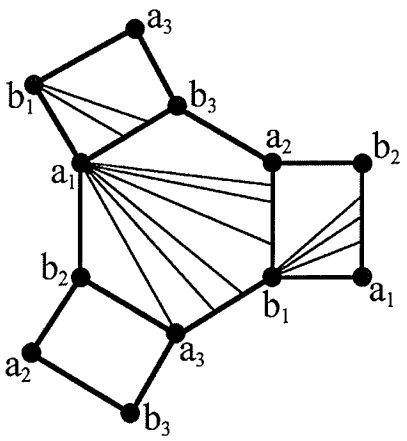

Figure 7.14: A pair of matched corner compatible Möbius bands

Figure 7.14 depicts a pair of matched corner compatible Möbius bands. Notice that the bands are corner compatible and matched for both common corners a_1 and b_1. As can be seen from Figure 7.14, there can be at most two matched corner compatible Möbius bands, the bands have to be compatible on the same corner and the third Möbius band must be flat to avoid chords crossing.

Proposition 7.2 *If graph G is projective planar, then there is a labelled embedding of $TK_{3,3}$ such that either Möbius bands are all flat, or one of the following* 3 *conditions hold:*

(i) non-flat Möbius bands are matched diagonal compatible;

(ii) at most two of the Möbius bands are non-flat and matched corner compatible on the same corner;

(iii) at most one of the Möbius bands is non-flat unmatchable and embeddable.

Proof. Since all three Möbius bands corresponding to three different quadragons are mutually crossing in the hexagon, we can embed only three diagonals in the hexagon as in Figure 7.13, or perpendicular chords incident on the same unique corner of the hexagon as in Figure 7.14. Otherwise all the perpendicular chords in the hexagon must correspond to the same Möbius band to avoid chords crossing. In any case, perpendicular chords that do not fit in the hexagon must be embedded without crossing in their corresponding quadragons. This gives at least one flat Möbius band for part (ii) and at least two flat Möbius bands for part (iii). ■

Corollary 7.1 *A projective planar embedding of G contains at most one unmatchable Möbius band with respect to an embedding of $TK_{3,3}$.*

Section 7.3.2 shows how to determine if an unmatchable Möbius band can be completed without chords crossing.

7.3.1 Diagonal and Corner Compatible Möbius Bands

Diagonal compatible Möbius bands are embeddable without chords crossing if they are matched on the appropriate diagonals. Referring to Figure 7.13, we

have just two triples of appropriate diagonals, namely, (a_1a_2, a_2a_3, a_3a_1) and (b_1b_2, b_2b_3, b_3b_1). All the other perpendicular chords different from a triple of diagonals must be placed without crossing in the quadragons. So, two triples of diagonals provide two particular cases of Möbius bands similar to the flat ones. In each particular case, we just embed the appropriate diagonals in the hexagon and check if the embedding of the other perpendicular chords in the quadragons causes any chords to cross. If crossing chords exists, the matched embedding of diagonal compatible Möbius bands is not possible.

It is convenient to consider a pair of matched corner compatible Möbius bands (see Figure 7.14) when a 3-face chord is embedded in the hexagon as in Figure 7.15.

Proposition 7.3 *A 3-face chord embedded in the hexagon makes one Möbius band flat and the other two must be matched corner compatible on the same corner, or one of the two is flat and the remaining one is embeddable (possibly, unmatchable).*

Proof. Suppose a 3-face chord is embedded in the hexagon as chord a_1b_1 in Figure 7.15. Then perpendicular chords of the Möbius band with sides b_2a_3 and b_3a_2 must fit in the quadragon not to cross the 3-face chord a_1b_1 in the hexagon. Perpendicular chords of the two other Möbius bands that are not incident on corner a_1 or b_1, must fit in their quadragons as well. The chords incident on corners a_1 and b_1 should not cross in the hexagon. ■

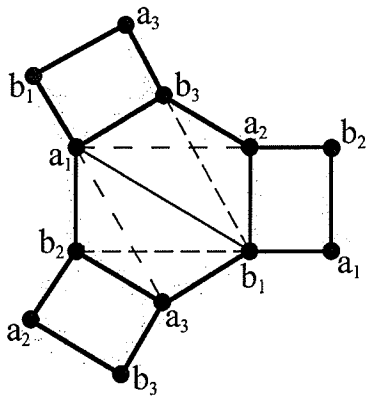

Figure 7.15: A 3-face chord in the hexagon

Notice that when embedded in the hexagon a 3-face chord becomes uniquely embeddable and is forced by 1-face chords as described in Section 7.2.3.

Two possibly corner compatible Möbius bands of Figure 7.14 have two common corners a_1 and b_1. The bands can be matched on either of the two common corners. So, we need to consider each corner separately to determine if we really have a pair of matched corner compatible Möbius bands. There are two cases to check, if the bands are matched depending on the common corner.

Suppose the bands of Figure 7.14 are matched on corner a_1. Clearly, perpendicular chords having both endpoints in interior vertices of the bands must be embedded in the quadragon, i.e. they become uniquely embeddable. The same is true for perpendicular chords having an endpoint in corner b_1. Therefore we need to place the perpendicular chords not incident on corner a_1 as

uniquely embeddable in the corresponding quadragon and this must not cause any chords to cross, to complete the embedding.

Now for perpendicular chords incident on corner a_1, we need to check if these perpendicular chords admit a bipartition such that one part of them is embedded in the hexagon and the other in the corresponding quadragon. Since any two perpendicular chords incident on the same corner never cross, we just need to check their possible crossings with other chords already embedded in the quadragon, for perpendicular and parallel chords.

Therefore each common corner a_1 and b_1 gives a special case for the completion of both Möbius bands. We can check the special cases of Möbius bands by using the results presented in the following section.

7.3.2 2-Face Chords in a Möbius Band

Finally, it is necessary to decide efficiently if there exists a bipartition of the parallel and perpendicular chords between the hexagon and quadragon for the only possible unmatchable Möbius band. If an embedding of $TK_{3,3}$ can be completed to G, then parallel and perpendicular chords of the unique unmatchable Möbius band admit a bipartition. Chords corresponding to one part of the bipartition are placed in the hexagon, and chords corresponding to the other part of the bipartition are placed in the quadragon and no two of

them cross. To determine efficiently if such a bipartition exists, we examine the possible crossings of chords in the Möbius band and a restriction of the conflict graph H to the corresponding chords.

The Structure of Crossing Chords

Lemma 7.3 *Given a Möbius band, two parallel chords of the same group cross in one face of the band if and only if they also cross in the other. A parallel chord crosses a perpendicular chord in one face of the band if and only if it also crosses in the other face.*

Proof. Each side of the Möbius band appears on the boundary of both faces. Clearly, the crossing of two parallel chords in one face implies that they cross in the other face as well (see Figure 7.16). The order of the endpoints of the parallel chords is just reversed in the other face. Similarly for a parallel and a perpendicular chord (see Figure 7.16). ■

Therefore, for either two parallel chords, or for a parallel and a perpendicular chord, we can determine if they are in conflict or not. Clearly, two chords in conflict must be placed in different faces of the Möbius band.

Lemma 7.4 *For a Möbius band, two perpendicular chords are*

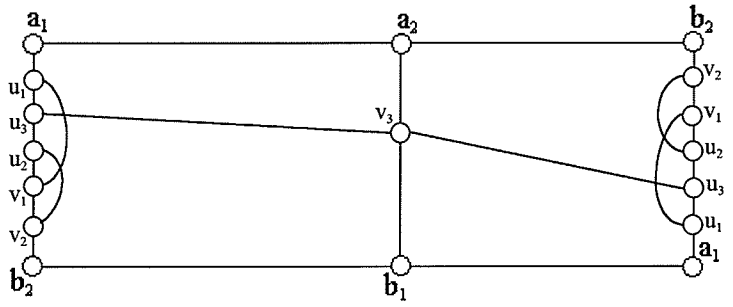

Figure 7.16: Chords in conflict

(i) disjoint if and only if they cross in one face and do not cross in the other face of the band.

(ii) adjacent if and only if they do not cross in either face of the Möbius band.

Proof. Part (i). For a Möbius band, two sides appear in a reversed order with respect to each other on the boundary of the two faces (see Figure 7.5). Therefore two disjoint perpendicular chords always cross in one face and do not cross in the other face as in Figure 7.17. Clearly, two perpendicular chords which cross in a face can not be adjacent.

Part (ii). As mentioned before, two adjacent chords never cross in a face. By part (i), two disjoint perpendicular chords cross in a face. Figure 7.18 gives an example of two adjacent perpendicular chords. ■

Corollary 7.2 *Two perpendicular chords corresponding to the same Möbius*

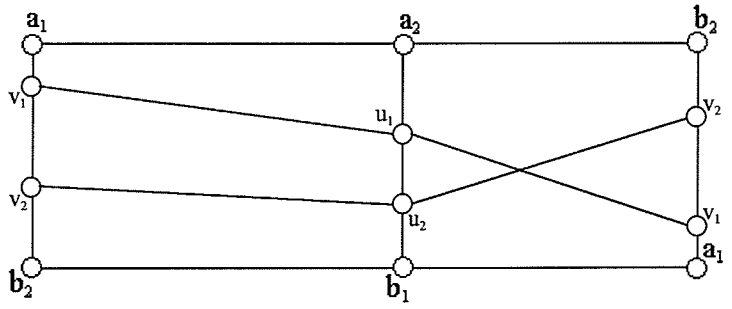

Figure 7.17: Disjoint perpendicular chords

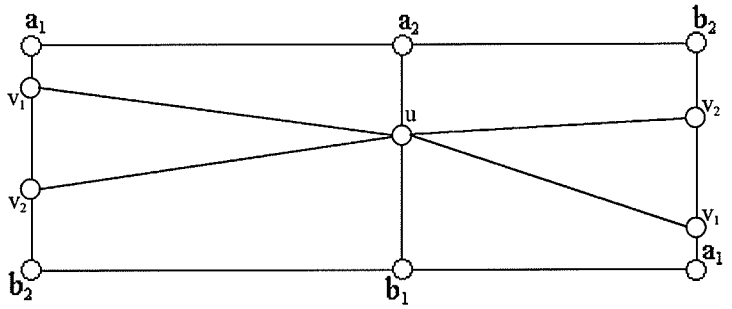

Figure 7.18: Adjacent perpendicular chords

band are never in conflict.

Definition 7.12 $M(H)$ denotes a restriction of the conflict graph H to chords corresponding to the Möbius band, i.e. $M(H)$ is a subgraph of H induced by vertices of the conflict graph which correspond to the chords of the Möbius band.

For a projective planar embedding of G, the conflict graph $M(H)$ must be bipartite. Otherwise, by Lemma 7.3, we can not draw the chords of the Möbius band without two parallel chords, or a parallel and a perpendicular chord crossing in one of the faces.

However even if the conflict graph $M(H)$ is bipartite, no two perpendicular chords are in conflict by Corollary 7.2 to Lemma 7.4. Therefore we need to check if there exists a bipartition of $M(H)$ corresponding to an embedding of the chords with no two perpendicular chords crossing in a face.

Components of the Restriction $M(H)$ of the Conflict Graph

Hereafter we assume the graph $M(H)$ is bipartite. To decide on the embeddibility of perpendicular chords corresponding to $M(H)$, we need to consider the connected components of $M(H)$. We can distinguish three types of connected components of $M(H)$ according to internal properties of the component.

Let C be a connected component of $M(H)$ with bipartition (A, B). Without loss of generality, assume $A \neq \emptyset$. Chords corresponding to A and B must be embedded in different faces of the Möbius band to avoid crossings of chords in conflict. Chords corresponding to A can be placed in either face. Therefore there are two different ways to embed C without two parallel chords or a parallel and a perpendicular chord crossing.

Definition 7.13 A component C is called *non-embeddable* if both possible embeddings of the component contains two perpendicular chords of A or B crossing in a face. Otherwise C is called *embeddable*.

Figure 7.19 shows an example of a non-embeddable component C of $M(H)$. Notice that the component of Figure 7.19 is bipartite.

Definition 7.14 A component C is called 1-*way embeddable* if one of two possible embeddings of the component contains two perpendicular chords crossing in a face but the other does not. A component C is called 2-*way embeddable* if both possible embeddings of the component contain no two perpendicular chords crossing.

Figure 7.20 shows an example of a 1-way embeddable component C of $M(H)$. Figure 7.21 shows an example of a 2-way embeddable component C of $M(H)$.

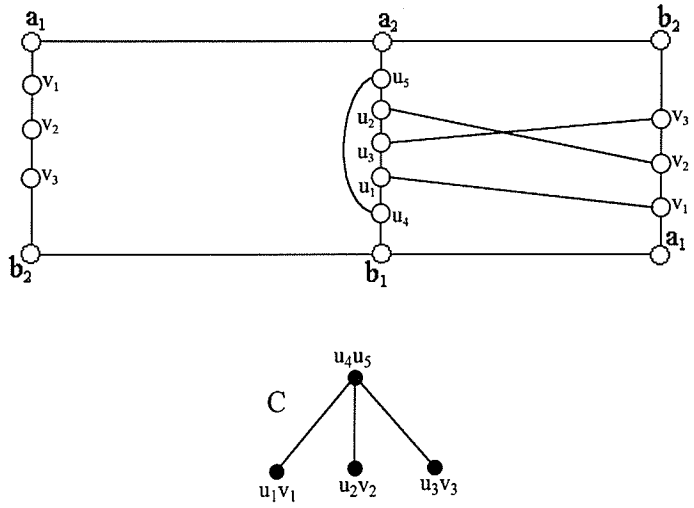

Figure 7.19: A non-embeddable component C of $M(H)$

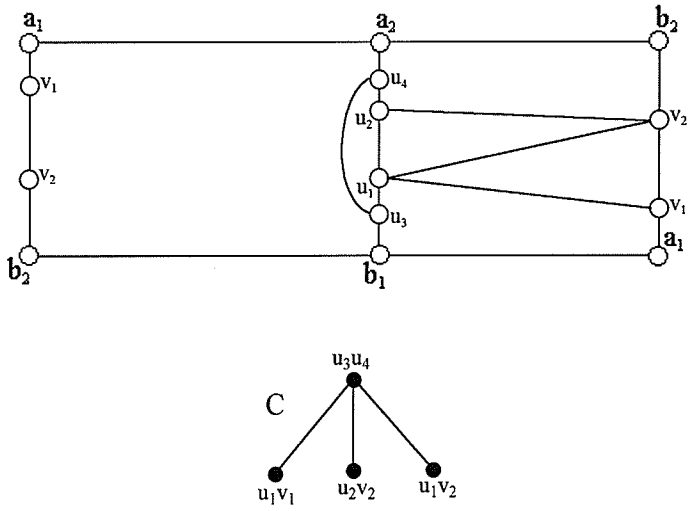

Figure 7.20: A 1-way embeddable component C of $M(H)$

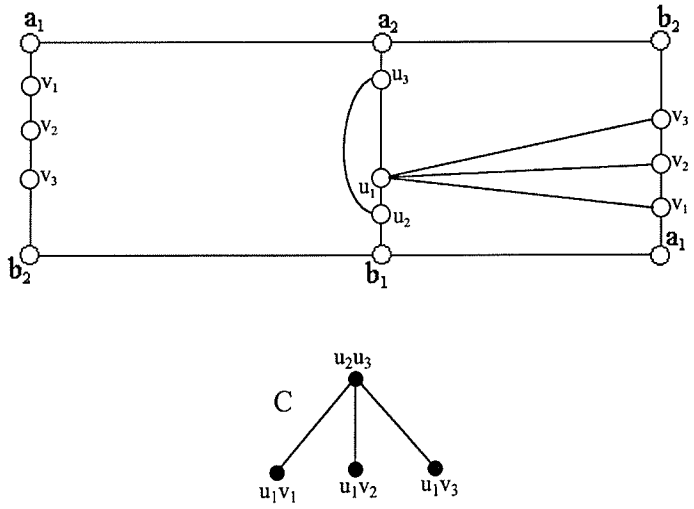

Figure 7.21: A 2-way embeddable component

Proposition 7.4 *An embeddable component C of $M(H)$ with bipartition (A, B), is 1-way embeddable if and only if A or B contains a pair of disjoint perpendicular chords. Otherwise C is 2-way embeddable.*

Proof. Necessity. By Lemma 7.4(i), if A or B contains a pair of disjoint perpendicular chords, the chords are crossing when embedded simultaneously in one of two faces of the Möbius band. Therefore all of A (or B) must be placed in the other face of the Möbius band and the component C is 1-way embeddable. Sufficiency. An embeddable component has its perpendicular chords split into two parts corresponding to A and B. Since C is 1-way embeddable, some perpendicular chords of the same part cross in one face but not in the other. By Lemma 7.4(i), two of these chords are disjoint. ■

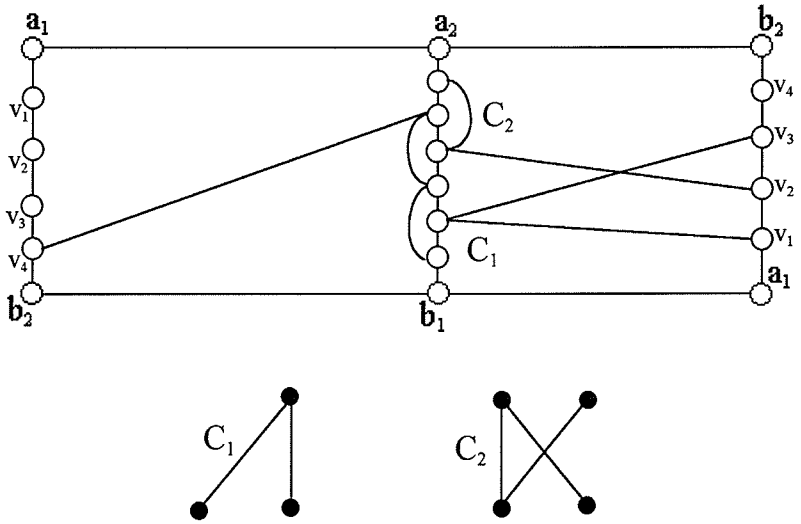

Figure 7.22: Embedding of component C_1 flips component C_2

Definition 7.15 Given an embedding of component C_i, we say the embedding of C_i *flips* an embedding of component C_j if the simultaneous embedding of C_i and C_j has corresponding perpendicular chords crossing. Component C_j is then embedded in the other way and called *flipped.*

Figure 7.22 shows an embedding of component C_1 that flips C_2. Notice that two components of Figure 7.22 are 2-way embeddable, and component C_2 has perpendicular chords in both faces of the Möbius band.

Definition 7.16 A component C of $M(H)$ is called *forced* if it is either

(i) a 1-way component, or

(ii) flipped by a 1-way component or by another forced component of $M(H)$.

Therefore a forced 2-way component is either uniquely embeddable, or not embeddable at all.

Proposition 7.5 *If G is projective planar, then there is an embedding of $TK_{3,3}$ such that:*

(i) $M(H)$ is bipartite;

(ii) all connected components of $M(H)$ are embeddable;

(iii) no two perpendicular chords from different 1-way or forced 2-way embeddable components are crossing.

Proof. Part (i) is a necessary condition to successfully embed the chords in conflict. Part (ii) is a necessary condition to have an embedding of a component without its own perpendicular chords crossing. Part (iii) is a necessary condition to successfully embed the uniquely embeddable components of $M(H)$. Figure 7.23 gives an example of two different 1-way embeddable components with perpendicular chords crossing. ■

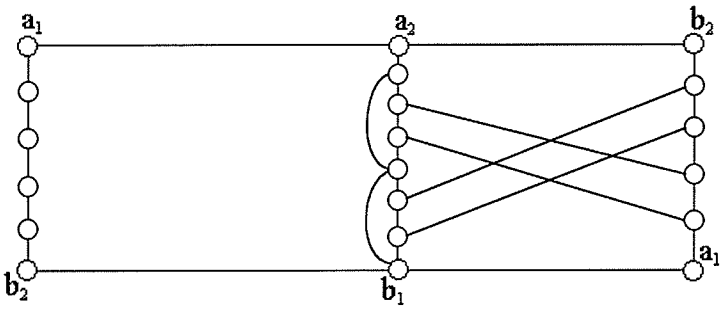

Figure 7.23: 1-way components with perpendicular chords crossing

2-Way Embeddable Components

It remains to choose an embedding for 2-way embeddable components of $M(H)$ that are not forced by 1-way embeddable components. By Proposition 7.4, all the perpendicular chords of each part of the bipartition of a 2-way embeddable component have a common endpoint (see Figure 7.21). We need to decide if it is possible to embed the 2-way embeddable components of $M(H)$ without their perpendicular chords crossing.

Let C_i and C_j be 2-way embeddable components of $M(H)$. There are 4 different ways to embed C_i and C_j simultaneously.

Definition 7.17 Components C_i and C_j are *independent* if perpendicular chords of C_i and C_j do not cross in any embedding of the components. Otherwise they are *dependent.*

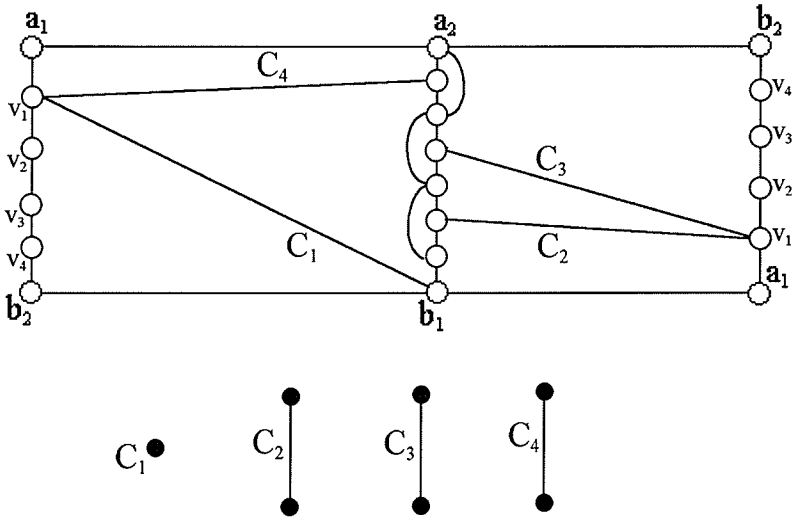

Figure 7.24: Independent 2-way components

Figure 7.24 shows a set of independent 2-way embeddable components and Figure 7.22 shows a pair of dependent 2-way embeddable components. By Lemma 7.4(ii), perpendicular chords of independent components are adjacent.

The following chapter explains how to use the structural results for different types of chords with respect to an embedding of $TK_{3,3}$ on the projective plane to devise a linear time embedding algorithm. The approach presented in Chapter 8 is similar to the Hopcroft-Tarjan planarity testing algorithm and is very complicated. The simple structural results of Chapter 7 can lead to a simpler projective planarity testing algorithm for graphs with a $K_{3,3}$-subdivision.

Chapter 8

The Projective Planarity Algorithm for Graphs Containing a $K_{3,3}$-Subdivision

This chapter provides a description of a linear time algorithm to check if there is an embedding in the projective plane for a graph containing a $K_{3,3}$-subdivision. The structural results of Sections 7.2 and 7.3 form the basis for the algorithm. The algorithm presented here can be considered as a generalization of the Hopcroft-Tarjan planarity algorithm.

First we describe an algorithm that decides if it is possible to embed chords

in the Möbius band and provides an embedding whenever one exists. The algorithm for the Möbius band is used in the projective planarity algorithm for graphs with a spanning $K_{3,3}$-subdivision. Clearly, some embeddings of parallel and perpendicular chords in the Möbius band can be forbidden by 1-face chords and forced uniquely embeddable chords (see Figures 7.5 and 7.9). This would make the parallel and perpendicular chords forced and uniquely embeddable. Also, referring to Figure 7.5, this can make a Möbius band restricted to shorter paths instead of whole sides of $TK_{3,3}$.

After the Möbius band algorithm, we present a general projective planarity algorithm given a spanning $K_{3,3}$-subdivision in the graph. Finally, a generalization for a non-spanning $K_{3,3}$-subdivision is described. This approach is similar to the Hopcroft-Tarjan algorithm of Chapter 3.

8.1 Embedding Chords in the Möbius Band

Consider a Möbius band determined by the hexagon interior and one of the quadragons. Without loss of generality, we can take the quadragon to be (a_1, b_2, a_2, b_1) (see Figure 7.4). In Figure 8.1 the quadragon is denoted by Q, and the hexagon is denoted by H. We assume that vertices of one side of the Möbius band are labelled and ordered as $v_p, v_{p-1}, \ldots, v_1$. Correspondingly, vertices of the other side of the Möbius band are labelled and ordered as

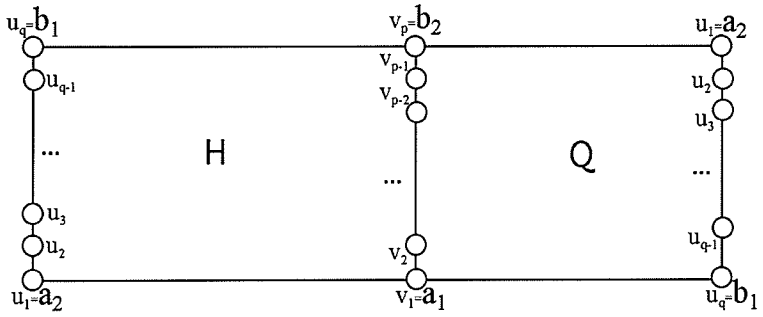

Figure 8.1: Möbius band labeling

$u_1, u_2, \ldots, u_q$ (see Figure 8.1).

8.1.1 Embedding Perpendicular Chords in the Möbius Band

First suppose we need only to decide if a set of perpendicular chords is embeddable into the two faces Q and H of the Möbius band (see Figure 8.1).

Perpendicular chords v_iu_j incident on the vertex v_i are considered in decreasing order of index $i = p, p-1, \ldots, 1$. We assume the adjacency list of v_i, $AdjList(v_i) = \{u_{i_1}, u_{i_2}, \ldots, u_{i_j}\}$, is ordered in increasing order of indices $i_1 < i_2 < \ldots < i_j$, and denote by $LastPerpPt(v_i) = u_{i_j}$, the last vertex in the adjacency list of v_i, $i = p,\ p-1,\ \ldots, 1$. We embed the chords $v_iu_{i_1}, v_iu_{i_2}, \ldots, v_iu_{i_j}$ incident to vertex v_i by considering them one by one.

By Lemma 7.4(i), any two disjoint perpendicular chords that cross in one face of the Möbius band do not cross in the other. To decide on the embedding of chord v_iu_j, we consider the next disjoint chord closest to v_iu_j defined as follows.

Definition 8.1 Let v_iu_j be a perpendicular chord. Denote by v_mu_k the perpendicular chord such that $m < i$, $k \neq j$, m is the largest index for which such a chord exists, and k is the smallest possible index. If such a chord v_mu_k exists, it is called the *next disjoint chord closest to* v_iu_j. Define $NextDisjPt(v_iu_j) := u_k$, if v_mu_k exists. If v_mu_k does not exist, define $NextDisjPt(v_iu_j) := u_{q+1}$, where u_{q+1} is a dummy vertex after u_q.

Given an embedding of perpendicular chords v_xu_y, $x = i+1, \ldots, p$, in faces Q and H of Figure 8.1, we denote by $HiPt_Q$ the vertex u_k such that $k = max\{y|$ chord $v_xu_y, x = i+1, \ldots, p,$ is embedded in $Q\}$. If the current chord to embed, v_iu_j, has $j < k$, it would cross a previously embedded chord in Q. Therefore v_iu_j can not be placed in the face Q.

Similarly, we denote by $HiPt_H$ the vertex u_k such that $k = min\{y|$ chord $v_xu_y, x = i+1, \ldots, p,$ is embedded in $H\}$. If the current chord to embed, v_iu_j, has $j > m$, it would cross a previously embedded chord in H. Therefore v_iu_j can not be placed in the face H.

The embedding algorithm places each chord v_iu_j, $i = p,\ p-1,\ \ldots, 1,$ into face

Q or H according to the highest available point $HiPt_Q$ and $HiPt_H$ in each face. If $u_j \geq HiPt_Q$ and $u_j \leq HiPt_H$, then chord v_iu_j can be placed in both faces without crossing previously embedded chords. We consider $NextDisjPt(v_iu_j)$ and $LastPerpPt(v_i)$ to decide on its embedding.

The auxiliary variables $NextHiPt_H$ and $NextHiPt_Q$ are used to calculate $HiPt_H$ and $HiPt_Q$ at the next iteration of algorithm for vertex v_{i-1}.

Algorithm 8.1 *Embedding Perpendicular Chords in the Möbius Band.*

Input: *A sequence of perpendicular chords for the Möbius band,* $AdjList(v_i)$, $i = p, p-1, \ldots, 1$, *sorted in increasing order of* u_j, $j = 1, 2, \ldots, q$

Output: *An embedding of the chords, or "not possible to embed"*

(1) Initialization:

for each vertex v_i, $i = p, p-1, \ldots, 1$, calculate $LastPerpPt(v_i)$

for each perpendicular chord v_iu_j calculate $NextDisjPt(v_iu_j)$

$HiPt_Q = u_1$

$HiPt_H = u_q$

(2) **for** every perpendicular chord v_iu_j

(in decreasing order of i and increasing order of j)

if $u_j < HiPt_Q$

chord v_iu_j *does not fit in face* Q

if $u_j > HiPt_H$

chord v_iu_j *does not fit in face* H

$NonProjective = true$

return

else

chord v_iu_j *can be placed in face* H

place v_iu_j in the hexagon H

$NextHiPt_H = u_j$

end if-else

else

chord v_iu_j *fits in face* Q

if $u_j > HiPt_H$

chord v_iu_j *does not fit in face* H

place v_iu_j in the quadragon Q

$NextHiPt_Q = u_j$

else

v_iu_j *can be placed in both face* H *and* Q

consider $NextDisjPt(v_iu_j)$

if $u_j > NextDisjPt(v_iu_j)$

place v_iu_j *in the hexagon*

it does not cross the next disjoint chord in the hexagon H

if $LastPerpPt(v_i) > HiPt_H$

the whole adjacency list of v_i *will not fit in the hexagon* H

place v_iu_j in the quadragon Q

$NextHiPt_Q = u_j$

else

the whole adjacency list of v_i *will fit in the hexagon* H

place v_iu_j in the hexagon H

$NextHiPt_H = u_j$

end if-else

else

we have $u_j < NextDisjPt(v_iu_j)$

place v_iu_j *in the quadragon* Q

place v_iu_j in the quadragon Q

$NextHiPt_Q = u_j$

end if-else

end if-else

end if-else

update the highest available points for faces Q *and* H

$HiPt_Q = NextHiPt_Q$

$HiPt_H = NextHiPt_H$

end for

Theorem 8.1 *If there is an embedding of perpendicular chords in faces* Q *and* H *of the Möbius band without crossing, then Algorithm 8.1 finds an embedding.*

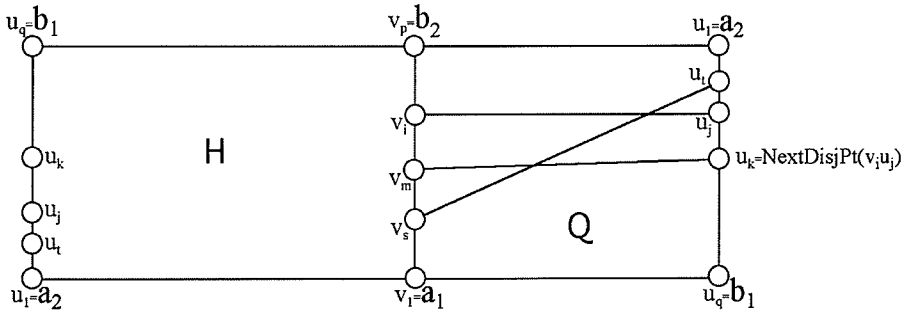

Figure 8.2: Configuration of perpendicular chords in the Möbius band

Proof. Referring to Algorithm 8.1, suppose we can embed chord v_iu_j in both faces Q and H without crossing previously embedded chords. First, assume $NextDisjPt(v_iu_j) = u_k > u_j$ and v_mu_k is the next disjoint chord closest to v_iu_j (see Figure 8.2). So v_iu_j and v_mu_k do not cross in face Q. Then any chord v_su_t, crossing v_iu_j from below in face Q, crosses v_mu_k as well when embedded simultaneously in face Q (see Figure 8.2). If v_su_t is embedded in Q, then v_iu_j and v_mu_k must be embedded in face H and they cross in H. Therefore v_su_t must be embedded in H to avoid v_iu_j and v_mu_k crossing in H. Just one of v_iu_j and v_mu_k can be embedded in H. Any perpendicular chord crossing v_iu_j in face Q also would cross v_mu_k in Q because v_mu_k is the disjoint perpendicular chord next to v_iu_j. Therefore we place v_iu_j in face Q.

The case $NextDisjPt(v_iu_j) = u_k < u_j$ is symmetric in face H. However if the last chord from the adjacency list $AdjList(v_i)$ of v_i will not fit in the hexagon H, without loss of generality we place v_iu_j in the quadragon Q to have $HiPt_H$

smaller for an embedding of the remaining chords. Otherwise we place $v_i u_j$ in the hexagon H by analogy with the previous case. ■

8.1.2 Embedding Bundles in the Möbius Band

Algorithm 8.1 can be generalized to find an embedding for bundles of parallel and perpendicular chords in the Möbius band. Each bundle corresponds to a non-embeddable, 1-way or 2-way embeddable component of the conflict graph $M(H)$ of Section 7.3.2. Therefore we call the bundles *non-embeddable*, 1-*way* or 2-*way embeddable* respectively.

For each bundle B, we can tell if it is a non-embeddable, 1-way or 2-way embeddable bundle by testing for disjoint perpendicular chords in each part of its bipartition. These are the internal properties of the bundle B and they can be detected in a process of bundle construction as a bipartite component.

If we know the bundles information in advance, we could easily decide on their embedding. For a 2-way bundle B, if it is possible, we embed the bundle to minimize both high available points $HiPt_Q$ and $HiPt_H$ for the next iteration of the algorithm. For example, if we embed the bundle of Figure 8.3 in the other way, both $HiPt_Q$ and $HiPt_H$ will be lower for the next iteration and the perpendicular chords of the bundle would cross chord e' in both faces Q and H. Therefore we place the bundle as it is.

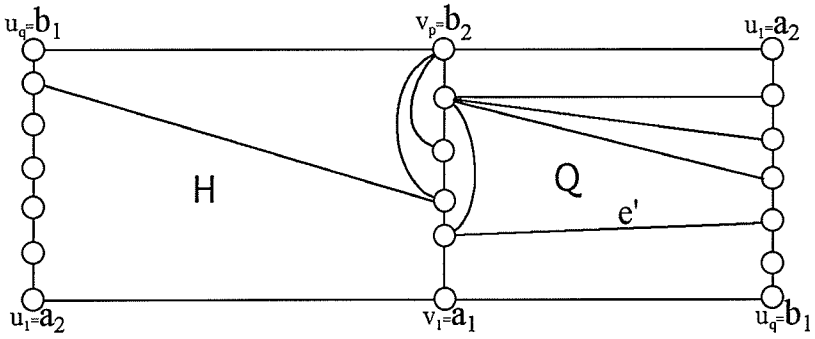

Figure 8.3: An optimal placement of a 2-way bundle

In the case when an optimal placement to minimize both $HiPt_Q$ and $HiPt_H$ is not possible, we consider the closest chord disjoint from a perpendicular chord of the bundle and try to place the bundle to avoid an intersection with the closest disjoint chord at the next iteration of the algorithm as in Algorithm 8.1.

The pure Hopcroft-Tarjan approach as in Chapter 3 does not assume that we know the bundles in advance – we construct and manipulate them during the embedding process. Therefore in this case we will need to know how to flip the bundles and how to determine when this is possible.

Interchanging Two Stars of Perpendicular Chords

Algorithm 8.1 constructs an embedding of a set of perpendicular chords in the Möbius band whenever such an embedding exists. However a set of perpen-

dicular chords can admit several different embeddings in the Möbius band and it can be necessary to add an embedding of parallel chords.

Having a simultaneous embedding of parallel and perpendicular chords in the Möbius band, the chords are in bipartite bundles corresponding to the connected components of the bipartite conflict graph. Since two perpendicular chords are never in conflict, the bundles are glued together by parallel chords. Without consideration of parallel chords, the embedding of perpendicular chords obtained by Algorithm 8.1 can be different from the embedding of the perpendicular chords in bundles. Therefore it can be necessary to modify the embedding of perpendicular chords of Algorithm 8.1 to add parallel chords to the embedding without crossing.

Suppose we have an embedding of perpendicular chords in faces H and Q of Figure 8.1. The vertices of side a_1b_2 are labelled in increasing order from a_1 to b_2, and the vertices of side a_2b_1 are labelled in increasing order from a_2 to b_1 (see Figure 8.1). The embedding of perpendicular chords is obtained after traversing side a_1b_2 from $v_p = b_2$ to $v_1 = a_1$. Let's assume that there is no parallel chord having both endpoints on the side a_1b_2 and each perpendicular chord itself is a bundle. The algorithm has to traverse the other side b_1a_2 of the Möbius band from $u_q = b_1$ to $u_1 = a_2$ to decide if we can complete an embedding with parallel chords having both endpoints on the side b_1a_2. The example of Figure 8.4 shows that it can be necessary to flip a perpendicular chord from one face to another to add the parallel chords without crossing.

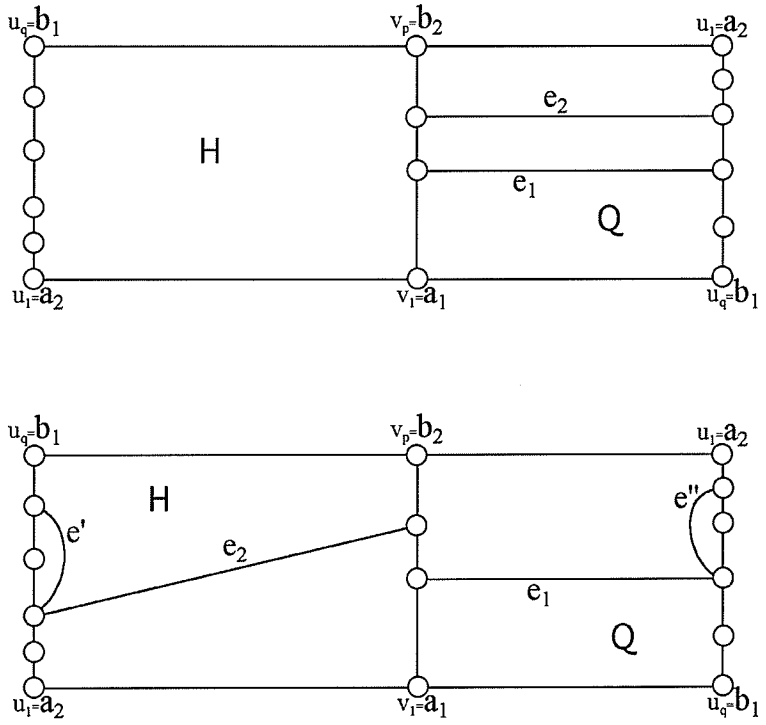

Figure 8.4: Perpendicular chord e_2 flipped by e_1 via parallel chords e', e''

Definition 8.2 A set of perpendicular chords having one endpoint in common is called a *perpendicular star*. The common endpoint is called the *star center*.

By Lemma 7.4, we can flip at most one perpendicular chord from any set of disjoint perpendicular chords embedded in face Q. The same is true for any set of disjoint perpendicular chords embedded in face H. Since perpendicular chords incident to the same vertex do not cross in either face H or Q, we can move at most one perpendicular star from face Q to face H. The same holds

for perpendicular chords embedded in face H.

Definition 8.3 For a perpendicular chord $e = v_i u_j$ embedded in face Q, its *closest upper chord* is a perpendicular chord $e' = v_{up}u_{up}$ embedded in the other face H such that $v_{up} > v_i$, v_{up} and u_{up} are the smallest possible (see Figure 8.5). If such a chord does not exist, we put $e' = u_q v_{p+1}$ the closest upper chord for e, where v_{p+1} is a dummy vertex right after v_p. Similarly, for a perpendicular chord $e = v_i u_j$ embedded in face Q, its *closest lower chord* is a perpendicular chord $e'' = v_{lo}u_{lo}$ embedded in the other face H such that $v_{lo} < v_i$, v_{lo} and u_{lo} are the biggest possible (see Figure 8.5). If such a chord does not exist, we put $e'' = u_1 v_0$ the closest upper chord for e, where v_0 is a dummy vertex right before v_1. Then the *flip interval* for chord $e = v_i u_j$ is defined as $[u_{lo}, u_{lo+1}, \ldots, u_{up}]$.

An initial flip interval for each perpendicular chord can be determined after running Algorithm 8.1 or its analogue. The flip interval bounds can be saved in an additional field for each perpendicular chord in the adjacency list.

If $u_{lo} \leq u_j \leq u_{up}$, then chord $e = v_i u_j$ can be flipped from face Q to face H without any problem. However if $u_j < u_{lo}$ or $u_j > u_{up}$, then chord e in face H would cross its closest upper or lower chord respectively. In this case we can initiate flipping of a perpendicular star with center u_{lo}, v_{lo}, u_{up} or v_{up} from face H to face Q to make the flip interval wider for e in H. For a perpendicular

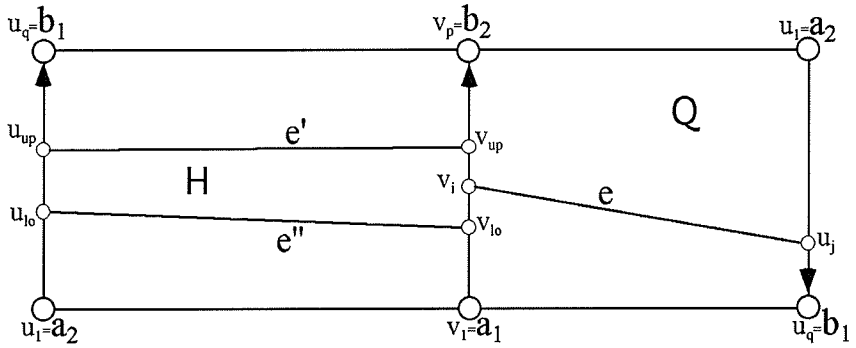

Figure 8.5: The flip interval for chord e

chord placed in face H, we can similarly define its flip interval in face Q to flip the chord from H to Q.

In summary, it can be necessary to move a perpendicular star from each face of the Möbius band to add parallel chords to an embedding without crossing. Moreover, moving a perpendicular star from a face of the Möbius band can force moving another perpendicular star from the other face of the Möbius band. The center of the last perpendicular star is defined by vertices like u_{lo}, v_{lo}, u_{up} or v_{up} of Figure 8.5.

Bundles in the Hopcroft-Tarjan Algorithm and in the Möbius Band

In the Hopcroft-Tarjan algorithm, the bundles are nested inside each other forming a stack. The innermost bundle is on top of the stack. Chords of each

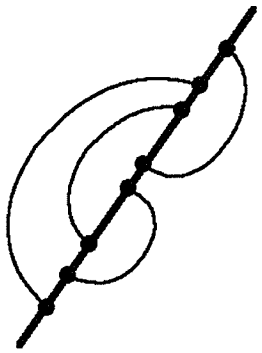

Figure 8.6: Consecutive chords in a bundle

bundle form a set of consecutive chords on left and right sides of the DFS-cycle or DFS-tree (see Figure 8.6).

When a chord uv is to be embedded, but conflicts with chords on both sides of the DFS-cycle or DFS-tree, then there is a possibility of switching the chords in the bundle to the other side that would permit chord uv to be embedded. This is only possible if the conflicting chords on both sides are not in the same bundle (see Figure 8.7).

After flipping chords of the innermost bundle, we merge the bundles containing conflicting chords and add uv to this bundle. Again, the new bundle consists of consecutive chords. Since the chords are always consecutive, we only need to store the first and last chords of the bundle on each side of the DFS-tree. Since switching the bundles can be done in constant time, the algorithm is linear. These properties are important for the linear time complexity of the

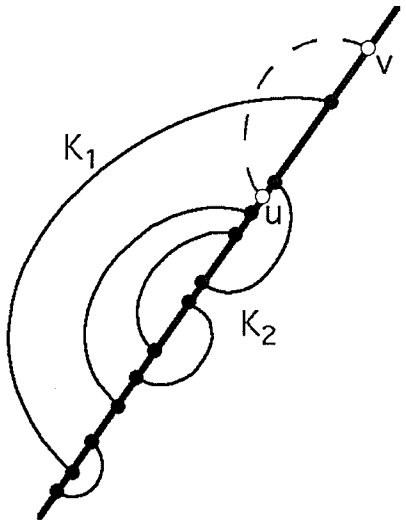

Figure 8.7: Flipping a bundle

Hopcroft-Tarjan algorithm and we can modify them to embed bundles in the Möbius band.

In a Möbius band there are two types of chords: parallel and perpendicular. Parallel chords behave exactly like chords in the Hopcroft-Tarjan algorithm. They form bundles that can be switched from one face to the other in the same way.

However there is an interaction between parallel and perpendicular chords that makes the algorithm more complicated. It is convenient to consider bundles that consist only of parallel chords first and then add perpendicular chords to them. If there are two disjoint perpendicular chords of the bundle embedded

in the same face, then it is not possible to flip the bundle by Lemma 7.4. Therefore to flip a bundle in the Möbius band, it is necessary that perpendicular chords of the bundle embedded in either face of the Möbius band have a common endpoint and form a perpendicular star. There can be a perpendicular star of the bundle embedded in the hexagon and a perpendicular star of the bundle embedded in the quadragon. There are four different cases of perpendicular stars in a bundle that can occur corresponding to four distinct combinations of two star centers (see Figure 8.8).

In the Möbius band bundles are not nested as in the Hopcroft-Tarjan Algorithm. If the bundles are nested in the Möbius band, only the outermost bundle can contain perpendicular chords, the inner bundles consist just of parallel chords, and it is possible to flip them as in the Hopcroft-Tarjan algorithm. Therefore we will assume that we have a Möbius bundle containing at least one perpendicular chord.

When we are trying to embed a parallel chord uv, if the chord uv must cross two disjoint perpendicular chords, then it is a forced chord, and we must embed uv in one face and the disjoint perpendicular chords in the other face without crossing. Therefore we assume that uv crosses one or more perpendicular chords that form a perpendicular star in a bundle.

If uv were to cross chords of the same bundle in the hexagon and quadragon, then flipping of the bundle would not help and we could not add uv to the em-

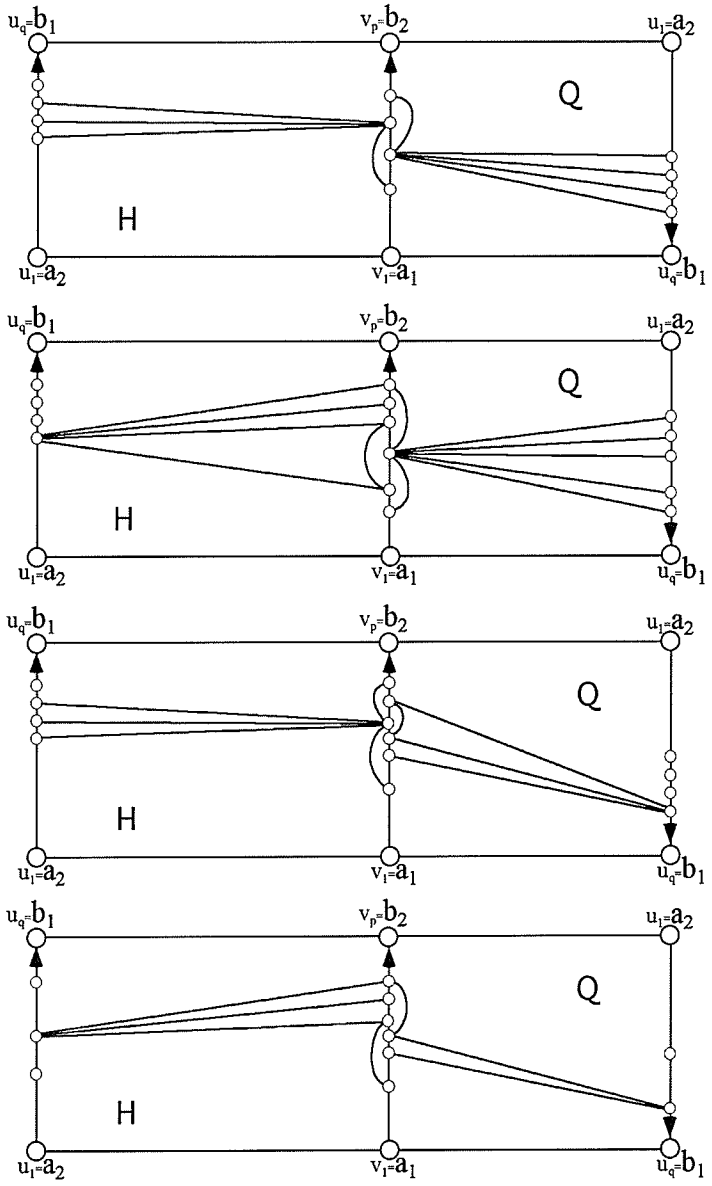

Figure 8.8: Four combinations of perpendicular stars in a bundle

bedding without crossing. Therefore the chords that uv crosses in the hexagon and quadragon are in different bundles. This is similar to the Hopcrof-Tarjan Algorithm. Then either uv crosses two perpendicular chords, or else a perpendicular and a parallel chord (see Figure 8.9). If vertex u is within a bundle, then we try to flip the bundle that contains u. If flipping the bundle that contains u is not possible without crossing, then we try to flip bundles that are the nearest to the bundle of u. It is possible to move just one perpendicular star out of each face of the Möbius band. As soon as we know the centers of two perpendicular stars that we need and can move from each face, the perpendicular chords that are not in the two perpendicular stars become fixed. Then it may become possible to embed other parallel chords without crossing or flipping any perpendicular chord not in the two perpendicular stars.

Consider flipping a perpendicular star with center A and possibly a perpendicular star with center B (see Figure 8.9). When the perpendicular chords incident to A are moved to the hexagon, they can cross the perpendicular chords immediately preceding the chords incident to B and those immediately following the chords incident to B. We look at the perpendicular chords immediately preceding and following the perpendicular star with center B to check if they would cross the perpendicular star with center A in the hexagon. If moving the perpendicular star with center A into the hexagon does not create any crossing, then the star with center A is moved into the hexagon and we proceed similarly with the star having center B. If there are conflicts, they must not be with two disjoint perpendicular chords. The only allowed con-

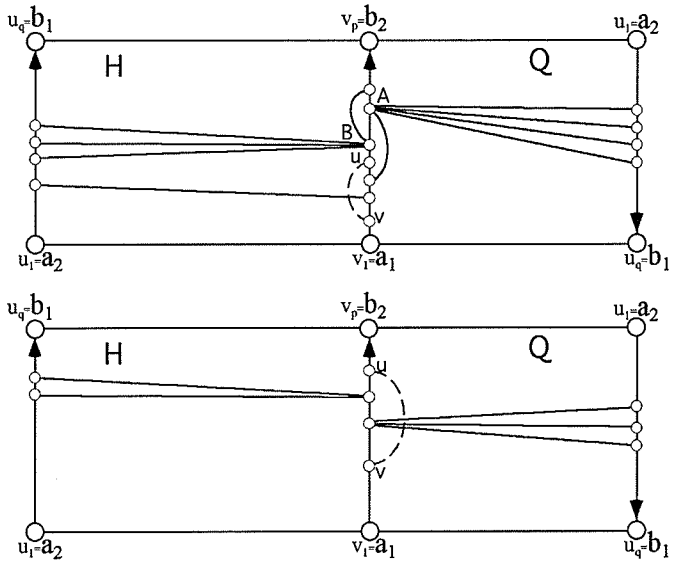

Figure 8.9: Flipping bundles to embed a parallel chord

flicts are with a perpendicular star, and then we need to move the conflicting perpendicular star too. This allows us to embed the chord uv. After flipping the two perpendicular stars, all the other perpendicular chords become rigid and it may become possible to complete the embedding without flipping any perpendicular chord.

Since chords in a perpendicular star are consecutive, flipping a perpendicular star takes a constant number of steps. It is necessary to know just the first and last chord of the star. The parallel chords in the bundle are also consecutive. If a bundle becomes rigid, we need to mark every chord in it as a rigid. This can happen at most once for each chord. This provides a linear running time for the entire algorithm.

8.2 Algorithm Given a Spanning $TK_{3,3}$

8.2.1 Data Structures

The graph G is represented by its adjacency list. Each edge of a linked list is marked either as an edge of $TK_{3,3}$ or as a chord of $TK_{3,3}$. The six vertices of $TK_{3,3}$ are marked as corners, the other vertices of $TK_{3,3}$ are marked as inner vertices. For each chord $e \in G \backslash ETK_{3,3}$ in the linked list, we store its *face index* set, the set of faces in which e can be embedded. The face index is

stored as a field for each entry of the linked list.

For each face F_i, $i = 0, 1, 2, 3$, we will use a linked list of chords to indicate the embedding of chords in the face as in the Hopcroft-Tarjan algorithm of Chapter 3. Thus we have four linked lists: $Face0Chords$ (the hexagon), $Face1Chords$, $Face2Chords$, and $Face3Chords$ (the quadragons). As the side between faces F_i and F_j, $i \neq j$, $i, j = 0, 1, 2, 3$, is traversed, the information from the $FaceiChords$ linked list and $FacejChords$ linked list is used later by the other sides on the face boundaries.

For bundles of quadragon chords, and parallel and perpendicular Möbius chords, a bundle stack is used to keep information about the bundles and their flips. A bundle is said to be *fixed* if it can not be flipped. Then all the chords in the same bundle are forced. Therefore the forced chords will appear as fixed bundles of quadragon chords in the $QuadragonPath$ procedure and as fixed bundles of parallel and perpendicular chords in $MobiusPath$ procedure.

8.2.2 DFS-numbering of the $K_{3,3}$-subdivision

We start with a $K_{3,3}$-subdivision $TK_{3,3}$. We are considering the case where $TK_{3,3}$ is spanning. $TK_{3,3}$ divides the projective plane into four faces – three quadragons and a hexagon. The faces interact through their common boundaries – the sides of the $TK_{3,3}$.

With the plane, there was a cycle C with two faces – the inside and outside. A sequential numbering of C which extends to a DFS-numbering of all of G is used to place the chords either inside C or outside C.

Now there is a spanning $TK_{3,3}$ instead of C, and four faces instead of two. However it is possible to treat the faces by considering them in pairs. We need a numbering of $TK_{3,3}$ that will extend to a DFS-numbering of all of G.

Consider the numbering schematically presented in Figure 8.10. Let the corners of $TK_{3,3}$ be a_1, a_2, a_3 and b_1, b_2, b_3. Denote a closed side of the $TK_{3,3}$ as $[a_2, b_3]$, indicating that the corners a_2 and b_3 are included in the side. Denote an open side as (b_3, a_3), indicating the inner vertices of the path. Start at a_2 and number it 1. Then number the vertices of $TK_{3,3}$ by taking the sides in the following sequence: $[a_2, b_3]$, (b_3, a_3), (a_2, b_1), $[a_3, b_1]$, (a_3, b_2), (b_1, a_1), $[a_1, b_2]$, (a_1, b_3), (b_2, a_2). It can be verified that every vertex of $TK_{3,3}$ is numbered, and that the corners are labelled once only, and in the correct sequence. Notice that this defines an ordering of the sides of $TK_{3,3}$ as nine directed paths $p_1, p_2, \ldots, p_9$. The inner vertices of each path p_i have numbers less than the vertices of any path p_j, when $j > i$, $i, j = 1, 2, \ldots, 9$.

Lemma 8.1 *With the numbering of Figure 8.10, the vertices of the facial boundaries of the three quadragons are numbered in increasing order.*

Proof. Consider the DFS-numbering of Figure 8.10. The boundaries of the

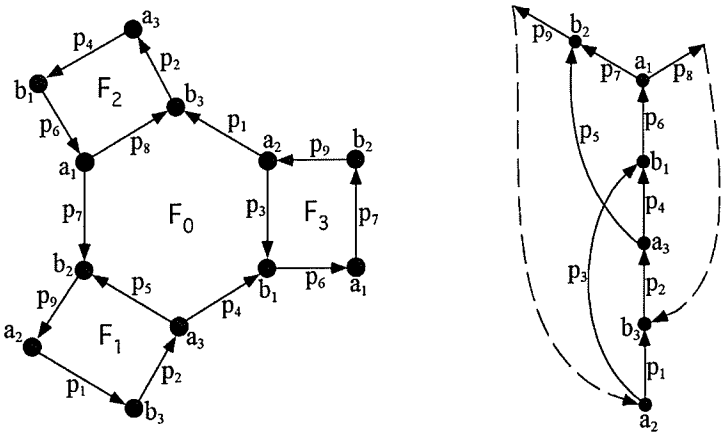

Figure 8.10: DFS-ordering of sides and face labelling of $TK_{3,3}$

quadragons have the numbering shown in Figure 8.11. ■

Having numbered the vertices of $TK_{3,3}$, we then traverse the sides of $TK_{3,3}$, in reverse order by decreasing DFS-numbers, exactly as the cycle C was traversed in reverse order by the DFS in Section 3.4, and assign a low point to each chord. In the case of a spanning $TK_{3,3}$, the low points are DFS-numbers of the adjacent vertices. We also order the adjacency lists of each vertex in order of increasing DFS-number. (We will come to the non-spanning case later.) We also determine which faces of the $TK_{3,3}$ each chord may be embedded in, and store this as a field in the adjacency lists.

We can determine the allowed faces for each chord in terms of the subscript numbers of the sides a_ib_j, $i, j = 1, 2, 3$, by using Proposition 7.1 and the face

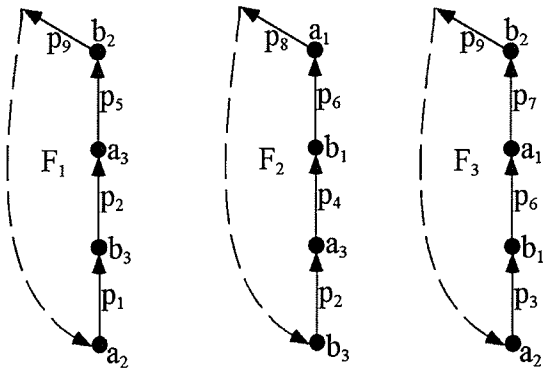

Figure 8.11: DFS-ordering of the quadragon facial boundaries

labelling of Figure 7.4. If we detect a 0-face chord, the embedding is not possible.

We are now ready to embed the chords. Notice that there are two kinds of sides in $TK_{3,3}$ – sides separating two quadragons and the Möbius band sides. Accordingly we have two procedures $QuadragonPath()$ and $MobiusPath()$ for embedding chords with an endpoint in one of these sides. We call *quadragon paths* the sides separating pairs of quadragons and *Möbius paths* the Möbius band sides.

8.2.3 The Quadragon Paths

While embedding chords on a quadragon path, there is a quadragon on each side of the path. Both quadragons have a DFS-numbering of their facial

boundaries as in Figure 8.11. Therefore they can be treated exactly as in the Hopcroft-Tarjan algorithm. To determine if a chord vu fits inside a face, we need only compare $DFNum[u]$ with the chord currently at the head of the list of chords for that face.

Algorithm 8.2 *Procedure* $QuadragonPath(a_i, b_i, Face1, Face2)$ *(embedding chords incident to a quadragon side* a_ib_i, $i = 1, 2, 3$*)*

Follow the quadragon path from a_i *to* b_i*, embedding each chord with an endpoint in this path.* $Face1$ *is to the "left" of the path,* $Face2$ *is to the "right"*

$v = a_i$

while (true)

  for each u adjacent to v do

    vu is a chord. Either u is above v in the $TK_{3,3}$*, or below; this can be detected by comparing DFS-numbers*

    if (u is above v in $TK_{3,3}$) **goto** L1; *the adj lists are ordered*

    if (uv is an edge of $TK_{3,3}$) **goto** L2; *uv is not a chord in this case otherwise uv is a chord with u below v in the* $TK_{3,3}$ *DFS-labelling*

    $Face1OK = true$, if uv's allowed faces include $Face1$

    $Face2OK = true$, if uv's allowed faces includes $Face2$

    if ($Face1OK$)

      uv may fit in $Face1$

if ($DFNum[u]$ fits inside $Face1$)

place uv inside $Face1$

else if ($Face2OK$)

uv may fit in $Face2$

if ($DFNum[u]$ fits inside $Face2$)

place uv inside $Face2$

else

uv is compatible with $Face1$ and $Face2$,

but does not fit inside either;

try switching sides

$k = QuadragonSwitchFaces(v, u, Face1, Face2)$

if ($k = 0$)

switching faces does not help

$NonProjective = true$

return

end if

otherwise switching faces made it possible to embed uv

if ($k = 1$) place uv inside $Face1$

else place uv inside $Face2$

end if $Face2OK$

otherwise uv is incompatible with $Face2$, and will not fit inside $Face1$;

there is still a possibility that it will fit in the hexagon if v is a corner

if ($Face1$ is the only allowed face for uv)

$NonProjective = true$

return

end if

otherwise this frond will be considered later by $MobiusPath()$

remove $Face1$ from uv's allowed faces

end if $Face1OK$

else if $(Face2OK)$

uv is not compatible with $Face1$*, but may fit in* $Face2$

if ($DFNum[u]$ fits inside $Face2$)

place uv inside $Face2$

else

uv will not fit inside $Face2$*. There is still a possibility that it will fit in the hexagon if v is a corner*

if ($Face2$ is the only allowed face for uv)

$NonProjective = true$

return

end if

otherwise this frond will be considered later by $MobiusPath()$

remove $Face2$ from uv's allowed faces

end if-else

end if $Face2OK$

L2: *go to the next u*

end for

L1: *prepare for the next v in the path*

if ($v = b_i$) return *all vertices of $a_i b_i$ done*

v = next vertex in the path

end while

end *QuadragonPath*

This procedure works very much like the Hopcroft-Tarjan embedding algorithm, except that we must indicate in which faces we are embedding the chords. It works because by Lemma 8.1 the facial boundaries of the quadragons are ordered by increasing DFS-numbers, and because the adjacency lists are ordered by the DFS-numbering.

We must always check each chord uv to see if it is compatible with $Face1$ or $Face2$ before embedding it. As before we have a linked list of chords for each face. *QuadragonSwitchFaces*() works very much like *SwitchSides*() of the Hopcroft-Tarjan algorithm except that there are more faces to consider. A stack of bundles of chords is required to indicate the chords which are grouped together. A bundle may be fixed by a 1-face or forced chord to become a bundle of forced quadragon chords.

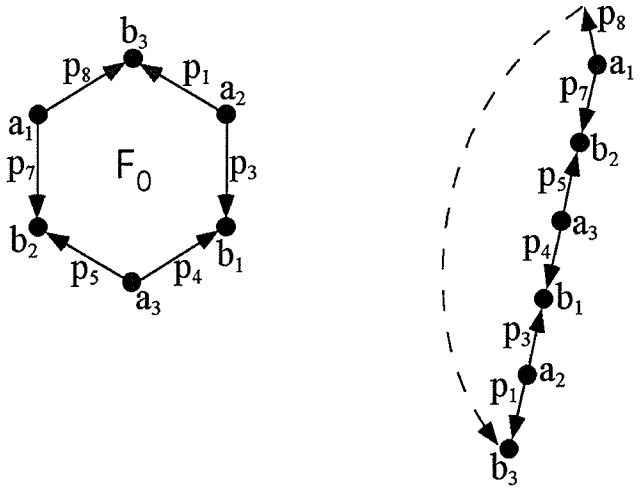

Figure 8.12: DFS-ordering of the hexagon facial boundary

8.2.4 The Möbius Paths

MobiusPath() is more complicated than *QuadragonPath*(). The reason is that the hexagon boundary is not ordered by increasing DFS-numbers. This is simply not possible if the ordering of the quadragon boundaries are increasing, because the projective plane is an unoriented surface. However, we do have an ordering of the hexagon's facial boundary with interesting properties. The boundary of the hexagon is shown in Figure 8.12.

Lemma 8.2 *Given the DFS-numbering of* $TK_{3,3}$ *of Figure 8.10, the hexagon boundary consists of paths of* $TK_{3,3}$ *taken in order of increasing subscript index, i.e.* p_1, p_3, p_4, p_5, p_7, p_8. *The paths of* $TK_{3,3}$ *are followed alternately*

according to decreasing and then increasing DFS-numbers, as we follow the hexagon boundary from the root vertex b_3 to its leaf on the path p_8.

Proof. See Figures 8.10 and 8.12. ■

Definition 8.4 We say that a path on which the DFS-numbers increase as we follow the path clockwise on the hexagon boundary is a *forward path.* A path in which the DFS-numbers decrease as we follow it clockwise on the hexagon boundary is a *reversed path.*

The algorithm needs to know which type of path it is following.

Algorithm 8.3 *Procedure* $MobiusPath(b_i, a_j, Face1, isReversed)$*:*
embed the chords incident on a hexagon side $b_i a_j$, $i, j = 1, 2, 3$, $i \neq j$

Follow the Möbius path from b_i *to* a_j*, embedding each chord with an endpoint in this path.* $isReversed$ *is true if this is a reversed path.* $Face1$ *(a quadragon) is to the "left" of the path,* $Face2$ *(=* $F0$*, the hexagon) is always to the "right".*
We always try to embed a chord in the quadragon $Face1$ *before the hexagon.*
We follow the path in order of decreasing DFS-numbers.
In $Face1$ *(a quadragon), this will be the usual order of the facial boundary;*
in $Face2$ *(the hexagon), this will be the alternating hexagon order.*
This means that from the point of view of the hexagon, we may be ascending

along the facial boundary, or descending.

$v = b_i$

$Face2 = F0$

while (true)

for each u adjacent to v do

vu is a chord. If it has already been embedded,

we can ignore it, and proceed to the next v

if (u has been embedded) goto L1; *the adjacency lists are ordered*

if (uv is an edge of $TK_{3,3}$) goto L2; *uv is not a chord in this case*

otherwise uv is a chord with u below v in the $TK_{3,3}$ DFS-labelling

$Face1OK = true$, if uv's allowed faces include $Face1$

$Face2OK = true$, if uv's allowed faces include $Face2$

if ($Face1OK$)

uv may fit in Face1

if ($DFNum[u]$ fits inside $Face1$)

place uv inside $Face1$

else if ($Face2OK$)

uv may fit inside the hexagon

if ($IsCompatible(u, v)$)

place uv inside $Face2$

else

uv is allowed in Face1 and Face2,

but it does not fit inside either. Try switching faces.

$k = MobiusSwitchFaces(v, u, Face1)$

if $(k = 0)$

Switching faces does not help.

$NonProjective = true$

return

end if

Otherwise switching faces made it possible to embed uv.

if $(k = 1)$ place uv inside $Face1$

else place uv inside $Face2$

end if $Face2OK$

Otherwise uv *is not allowed in* $Face2$*, and will not fit inside* $Face1$.

There is still a possibility that it will fit into another quadragon,

if v *is a corner.*

if ($Face1$ is the only allowed face for uv)

$NonProjective = true$

return

end if

otherwise this chord will be considered later by $QuadragonPath()$

remove $Face1$ from uv's allowed faces

end if $Face1OK$

else if $(Face2OK)$

uv *is not allowed in* $Face1$*, but may fit in* $Face2$

if $(IsCompatible(u, v))$

place uv inside $Face2$

else

uv will not fit inside Face2. There is still a possibility that it will fit in a quadragon if v is a corner

if ($Face2$ is the only allowed face for uv)

$NonProjective = true$

return

end if

otherwise this frond will be considered later by QuadragonPath()

remove $Face2$ from uv's allowed faces

end if-else

end if $Face2OK$

L2: *go to the next u*

end for

L1: *prepare for the next v in the path*

if ($v = a_j$) **return** *all vertices of* $b_i a_j$ *done*

v = next vertex in the path

end while

end $MobiusPath$

$MobiusPath()$ must be coded separately from $QuadragonPath()$ because of the hexagon boundary order. A procedure $IsCompatible(u, v)$ is used to de-

termine whether a chord uv can be placed in the hexagon. It must determine whether u and v are in forward or reversed paths of the hexagon boundary, and decide on how to compare uv with the chord currently at the head of $Face0Fronds$, the list of chords embedded in the hexagon. Then it decides whether the chord will fit into the hexagon. The handling of the linked lists of bundles of chords is also different for $MobiusPath()$, because sometimes we are ascending the facial boundary, and sometimes descending.

If a chord does not fit into the hexagon, then we call $MobiusSwitchFaces()$. This is where a bundle of one or more chords may be switched between the hexagon and its adjacent quadragon. Section 8.1 explains how to decide on an embedding of parallel and perpendicular chords into the Möbius band in more detail. The algorithms of Section 8.1 must be inserted into the $MobiusPath()$ procedure.

8.2.5 Möbius Bands and the General Algorithm

Proposition 7.2 and Corollary 7.1 simplify the embedding along the Möbius paths by reduction to the special cases. In general case, we run $MobiusPath()$ procedures assuming at most one non-flat embeddable Möbius band. Otherwise the general case run of the algorithm will determine the number of non-flat Möbius bands and their common corners in case of a pair of corner-compatible Möbius bands. If all the Möbius bands are flat or only one is non-flat and

embeddable, we obtain an embedding.

If there are three non-flat Möbius bands, we try two cases of the diagonal compatible bands by making the diagonals uniquely embeddable in the hexagon, and all the other perpendicular chords, uniquely embeddable in the corresponding quadragons. If there are two non-flat Möbius bands, we try two cases of the corner compatible bands by making the perpendicular chords not incident on the corresponding corner uniquely embeddable in the quadragons. Finally, if we have just one non-flat Möbius band, the first run of the algorithm will provide an embedding, if one exists. The general embedding algorithm can now be described as follows.

Algorithm 8.4 *Given a spanning $TK_{3,3}$, embed its chords in the projective plane*

(1) Construct the DFS-numbering of $TK_{3,3}$

(2) Assign the allowed faces for each chord

if $(NonProjective)$ **return** *there was a 0-face chord*

Now follow the paths of $TK_{3,3}$ in reverse order, embedding chords as we go

(3) $QuadragonPath(a_2, b_2, F_1, F_3)$ *path* p_9

if $(NonProjective)$ **return**

(4) $MobiusPath(b_3, a_1, F_2, false)$ *path* p_8

if $(NonProjective)$ **return**

(5) $MobiusPath(b_2, a_1, F_3, true)$ *path* p_7

if $(NonProjective)$ **return**

(6) $QuadragonPath(a_1, b_1, F_2, F_3)$ *path* p_6

if $(NonProjective)$ **return**

(7) $MobiusPath(b_2, a_3, F_1, false)$ *path* p_5

if $(NonProjective)$ **return**

(8) $MobiusPath(b_1, a_3, F_2, true)$ *path* p_4

if $(NonProjective)$ **return**

(9) $MobiusPath(b_1, a_2, F_3, false)$ *path* p_3

if $(NonProjective)$ **return**

(10) $QuadragonPath(a_3, b_3, F_1, F_2)$ *path* p_2

if $(NonProjective)$ **return**

(11) $MobiusPath(b_3, a_2, F_1, true)$ *path* p_1

if $(NonProjective)$ **return**

The embedding is now complete

(12) Calculate a rotation system

return

8.2.6 Assigning a Rotation System to the Embedding

When Algorithm 8.4 completes an embedding of $TK_{3,3}$ to an embedding of the whole graph G on the projective plane, we still need to calculate a rotation sys-

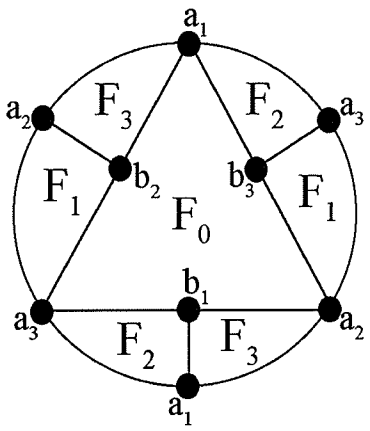

Figure 8.13: The embedding of $TK_{3,3}$ used to calculate a rotation system

tem of the embedding. We know the assignment of chords to faces of $TK_{3,3}$. The cyclically ordered adjacency lists for an embedding of G can be constructed from the labelled embedding of $TK_{3,3}$, and the assignment of chords to the faces. Also we can use the information from the linked lists $Face0Chords$, $Face1Chords$, $Face2Chords$ and $Face3Chords$ calculated during the previous computations.

A rotation system for a projective planar embedding requires assigning a signature of "+1" or "-1" to each edge of the embedding. The value of "-1" is assigned to the edges crossing the projective plane boundary. All the remaining edges are assigned the value of "+1".

The embedding of Figure 8.13 is used to assign signature values to the edges of an embedding. This embedding of $TK_{3,3}$ is a continuous transformation of the

embedding of Figure 7.2 used by the algorithm. The value of "-1" is assigned to perpendicular chords of the Möbius bands embedded in the quadragons and to the 1-face chords having endpoints on the opposite quadragon sides a_ib_i and a_jb_j, $i, j = 1, 2, 3$, $i \neq j$, or on the quadragon sides incident to the same corner a_i, $i = 1, 2, 3$. In Figure 8.13, only the quadragons are separated into two parts by the projective plane boundary on their diagonal with respect to a pair of opposite corners. Therefore the only chords that must cross the projective plane boundary will be embedded in a quadragon and have endpoints on the quadragon opposite or corresponding adjacent sides. All other chords can be drawn without crossing the boundary and are assigned the value of "+1".

8.3 Generalization for a Non-Spanning $TK_{3,3}$

The case of a non-spanning $TK_{3,3}$ is a reasonably straight-forward extension of the spanning case. We first assign the DFS-numbering of the $TK_{3,3}$ exactly as in the spanning case. Then we follow the paths of the $TK_{3,3}$, in reverse order. While visiting a vertex v, if we encounter a vertex u which is not part of the $TK_{3,3}$ and that has not been numbered, we call a recursive procedure $LowPtDFS()$ at vertex u. $LowPtDFS(u)$ proceeds exactly as in the Hopcroft-Tarjan algorithm. The entire DFS-tree constructed by $LowPtDFS(u)$ must fit inside one face of the $TK_{3,3}$. We calculate the allowed faces for this tree as $LowPtDFS(u)$ proceeds by checking the attachments of

each tree leaf, and store the tree's allowed faces in the adjacency list of v when $LowPtDFS(u)$ returns. Thus we can treat the entire tree recursively as a chord at v. When we have followed all nine paths of $TK_{3,3}$, the entire graph G has been numbered, and the allowed faces have been calculated for every chord and DFS-tree which attaches to the $TK_{3,3}$.

Then the algorithm proceeds to embed the chords and back edges in the linked lists. The same sequence of calls to $QuadragonPath()$ and $MobiusPath()$ as in Algorithm 8.4 above will embed the graph. The only change is that we must add some additional statements to each of $QuadragonPath()$ and $MobiusPath()$. The additional statements for $QuadragonPath()$ are as follows. The statements for $MobiusPath()$ are similar.

Algorithm 8.5 *Extension of QuadragonPath() procedure for non-spanning* $TK_{3,3}$

for each u adjacent to v do
  if (u has not been visited)
    v is the root of a DFS-tree T attaching to the $TK_{3,3}$.
    $Face1OK = true$, if T is allowed in $Face1$
    $Face2OK = true$, if T is allowed in $Face2$
    if ($Face1OK$)
      $EmbedBranch(u, Face1)$
      if (successful) goto L2;

Otherwise try $Face2$

if ($Face2OK$)

$EmbedBranch(u, Face2)$

if (successful) goto L2;

Otherwise T *will not fit inside* $Face2$*. Try switching faces.*

$k = QuadragonSwitchFaces(v, u, Face1, Face2)$;

if ($k = 0$)

$NonProjective = true$

return

end if

if ($k = 1$) $EmbedBranch(u, Face1)$

else $EmbedBranch(u, Face2)$

end if $Face2OK$

Otherwise T *is not allowed in* $Face2$*, and will not fit inside* $Face1$*.*

There is still a possibility that it will fit into the hexagon

if v *is a corner.*

if ($Face1$ is the only allowed face for T)

$NonProjective = true$

return

end if

Otherwise T *will be considered later by* $MobiusPath()$*.*

remove $Face1$ from T's allowed faces

end if $Face1OK$

else if ($Face2OK$)

$EmbedBranch(u, Face2)$

if (successful) goto L2;

Otherwise T is not allowed in $Face2$, and will not fit inside.

There is still a possibility that it will fit into the hexagon

if v is a corner.

if ($Face2$ is the only allowed face for T)

$NonProjective = true$

return

end if

Otherwise T will be considered later by $MobiusPath()$.

remove $Face2$ from T's allowed faces

end if $Face2OK$

Otherwise T will be considered later by $MobiusPath()$.

else

vu is a chord of $TK_{3,3}$. Previous code to embed

chords at vertex v by $QuadragonPath()$ goes here.

...

...

...

end if-else

L2: *go to the next u*

end for

The procedure $EmbedBranch(u, Face)$ is similar to the procedure used for the plane. It follows a DFS-tree to a leaf node, finds the initial cycle C, and then returns along the tree as the recursion unwinds, and places chords in one of two linked lists, one for the inside of C, the other for the outside. All fronds must fit inside the given face of $TK_{3,3}$. The alternating hexagon boundary DFS-ordering must be used to embed a branch in the hexagon. The implementation can be simplified by storing an additional array $HexagonNum[v]$, giving a consecutive numbering of the vertices on the hexagon boundary.

8.4 Analysis and Complexity of the Algorithm

A final algorithm can be constructed from Algorithms 8.1-8.3 and 8.5 for all cases of flat, compatible and unmatchable Möbius bands. It can be done similar to Algorithm 8.4. The two possible cases of three matched diagonal compatible Möbius bands, and two possible cases of two matched corner compatible Möbius bands can be programmed separately. The entire algorithm treats the case of three flat Möbius bands, and the case of two flat and one unmatchable Möbius bands before the special cases.

The algorithm is similar to the Hopcroft-Tarjan planarity algorithm of Chapter

3. It consists of a constant number of DFS procedures. Each DFS procedure runs in linear time.

The algorithm considers in turn all six labelled embeddings of $TK_{3,3}$ from Section 7.2.1. The DFS-numbering of $TK_{3,3}$ and DFS-tree generation in $G \backslash TK_{3,3}$ require $O(n+m)$ operations. The embedding part uses information about edges of G not in $TK_{3,3}$ and not in the spanning DFS-trees of $G \backslash TK_{3,3}$. It consists of a sequence of linked list manipulations, adding or deleting information from the adjacency lists fields. The adding or deleting of an element requires a constant time. The total number of entries in the linked lists is $O(n+m)$. Therefore the entire algorithm runs in $O(n+m)$ time. Since $m \leq 3n-3$ for a projective planar graph, the algorithm requires $O(n)$ operations. Since we use a constant number of fields for each entry of the adjacency list, the memory storage is also $O(n)$.

The algorithm can be followed by a procedure to convert the linked lists $FaceiChords$, $i = 0, 1, 2, 3$, to a rotation system for an actual embedding of the graph as described in Section 8.3.6.

Since we have to check all six labelled embeddings of $TK_{3,3}$ to tell if a graph G is non-projective planar, the algorithm can be difficult to modify to find a minimal non-projective planar subgraph in G. In addition, a distinct non-projective planar obstruction can appear for each labelled embedding of $TK_{3,3}$.

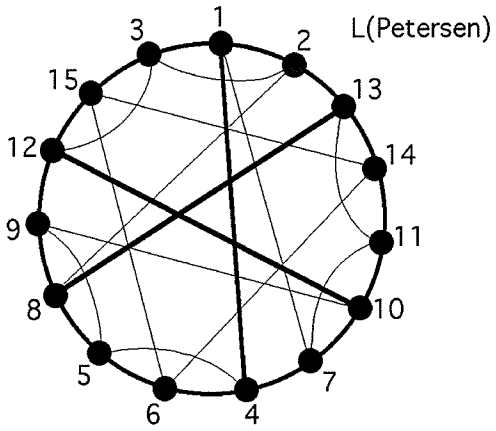

Figure 8.14: The line graph of the Petersen graph

8.5 Examples

The program implementing this algorithm is currently in the debugging phase. One example in which it embedded the line graph of the Petersen graph is illustrated.

Figure 8.14 shows the line graph of the Petersen graph. This is a 4-regular graph with fifteen vertices and thirty edges. The $K_{3,3}$-subdivision is shown by bold lines, the chords of $TK_{3,3}$ are shown by thin lines. The corners of $TK_{3,3}$ are labelled $a_1 = 1, a_2 = 10, a_3 = 8$ and $b_1 = 4, b_2 = 12, b_3 = 13$.

The program tried to complete all six labelled embeddings of the $K_{3,3}$-subdivision with twelve chords. Three labelled embeddings of $TK_{3,3}$ had a 0-face chord

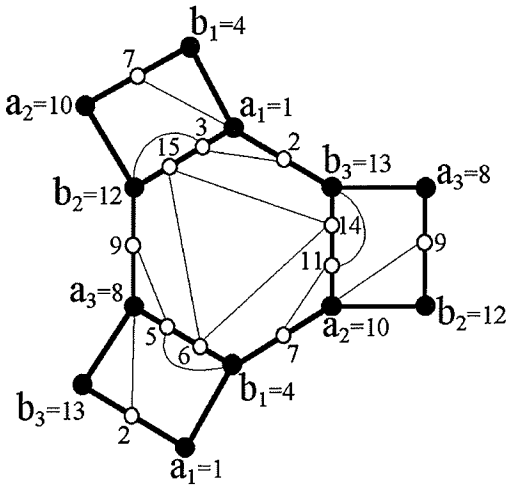

Figure 8.15: A projective planar embedding example

and therefore were rejected by the program.

Two embeddings of $TK_{3,3}$ gave actual embeddings of the line graph of the Petersen graph on the projective plane. The embeddings are shown in Figure 8.15 and Figure 8.16.

Finally, one labelled embedding had both embeddings of the 2-face chord with endpoints 11 and 13 crossed by 1-face chords in two corresponding quadragons as in Figure 8.17.

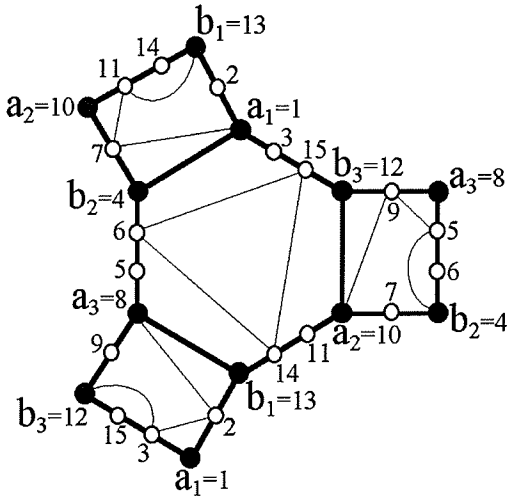

Figure 8.16: A projective planar embedding example

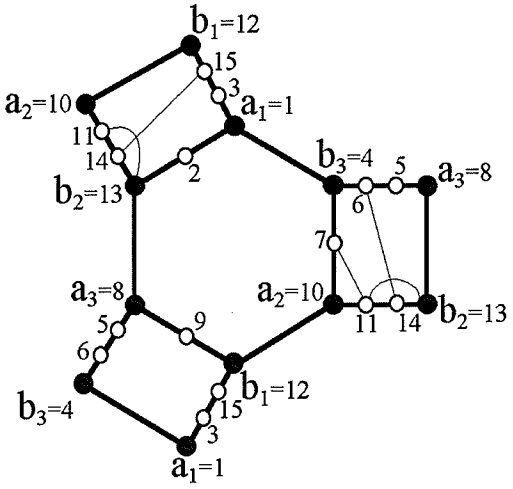

Figure 8.17: Non-projective configuration example

Chapter 9

Torus Embeddings of Graphs Containing K_5–Subdivisions

In this chapter, we describe a partial algorithm to embed graphs in the torus. The algorithm checks if a graph G containing a K_5-subdivision is toroidal or if it contains a $K_{3,3}$-subdivision. The algorithm has linear time complexity and is similar to the algorithm for the projective plane presented in Section 7.1. It can be implemented in a straightforward way to return an actual embedding of the graph.

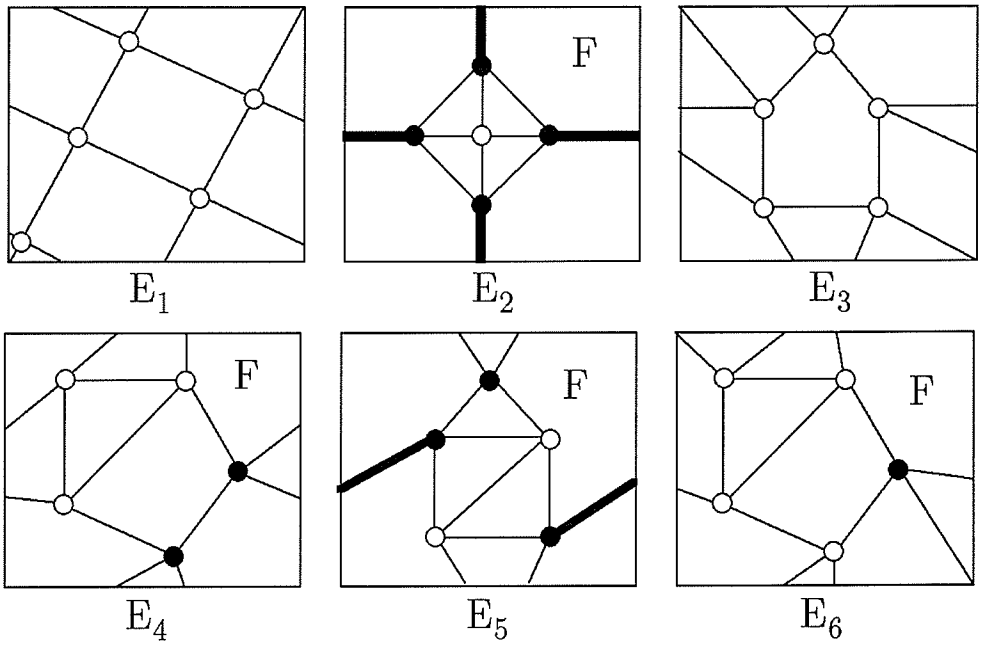

Figure 9.1: The embeddings of K_5 on the torus

9.1 Embedding K_5-Subdivisions on the Torus and Planar Side Components

We begin with the six embeddings of K_5 on the torus, shown as $E_1, \ldots, E_6$ in Figure 9.1. Some embeddings have one face whose boundary contains a repeated vertex or repeated edge. Such a face is labelled F in the diagram. Vertices which are repeated on the boundary of F are shaded black. Repeated edges are drawn with thicker lines.

Let G be a non-planar graph with a K_5-subdivision TK_5 and no short cut or 3-corner vertex of TK_5 in G. Notice that a side component of TK_5 in G can contain a subdivision of $K_{3,3}$. Therefore G can contain a $TK_{3,3}$ as well. The following propositions and theorem provide a characterization of toroidality for such graphs.

Proposition 9.1 *Let G be a 2-connected non-planar graph with a K_5-subdivision TK_5 and no short cut or 3-corner vertex of TK_5 in G. If G is toroidal, then at most one augmented side component of TK_5 is non-planar.*

Proof. Let G be embedded on the torus. Consider the embeddings of K_5 on the torus $E_1, \ldots, E_6$ of Figure 9.1. TK_5 must be embedded in one of these configurations. Let H be any side component with corners a and b. The vertices of H cannot be adjacent to any part of TK_5, except those vertices on the $\{a, b\}$-side. We show that either $H + ab$ is planar, or else all other augmented side components are planar.

Case 1. TK_5 is embedded as E_1 or E_3 of Figure 9.1.

E_1 and E_3 have the property that each vertex appears at most once on the boundary of any face. A side $\{a, b\}$ appears on the common boundary of two faces, say F_1 and F_2. Vertices a and b may also appear as non-consecutive vertices on the boundary of a third face. We proceed as in Theorem 7.1. Any portion of H embedded in a third face can be moved to F_1 or F_2, so that H

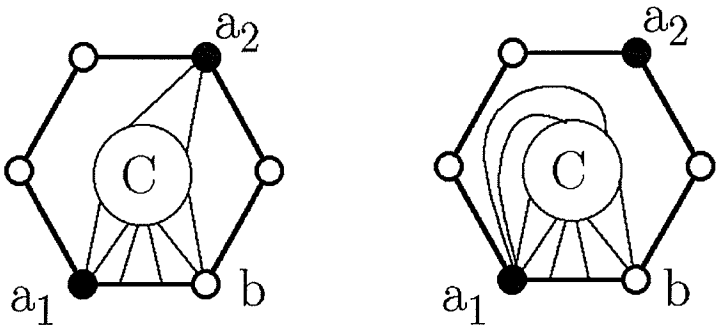

Figure 9.2: A face with one repeated vertex

can always be embedded in an open disk contained in $F_1 \cup F_2$, with a and b on the outer face of H. Lemma 6.1 implies that $H + ab$ is planar.

Case 2. TK_5 is embedded as E_6 of Figure 9.1.

The boundary of the face F contains one vertex repeated twice as in Figure 9.2. Without loss of generality, we can assume that a is the repeated corner, and b is adjacent to a on the boundary of F. Otherwise $H + ab$ would be planar, as in Case 1. Let C be a part of H embedded in the interior of F. Let a_1 and a_2 be the two occurrences of a on the boundary of F, and let a_1 be adjacent to b on the facial foundary. The edges from a_2 to vertices $v \in C$ can be replaced by edges from a_1 to v, as indicated in the diagram. This gives a planar embedding of H with a and b on the outer face. Hence $H + ab$ is planar.

Case 3. TK_5 is embedded as E_4 of Figure 9.1.

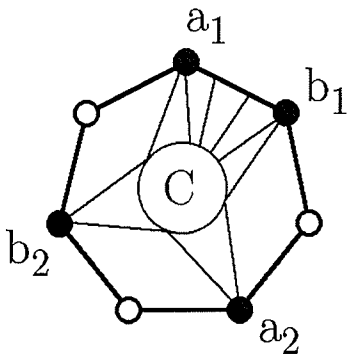

Figure 9.3: A face with two repeated vertices

The boundary of the face F of E_4 has two vertices repeated twice as in Figure 9.3. Without loss of generality, we can take one of them to be a. Let its two occurrences on the facial boundary be a_1 and a_2. If b is not the other repeated corner, we can proceed as in Case 2 and $H + ab$ is planar. Hence, we can assume that b is also repeated. Let its two occurrences be b_1 and b_2, where b_1 is adjacent to a_1 on the facial boundary. Let C be the portion of H embedded inside F. Each of a_2 and b_2 must be adjacent to one or more vertices of C, or else we can proceed as in Case 2, and $H+ab$ is planar. Having embedded C as shown in Figure 9.3, all faces of the embedding now have no repeated vertices on their boundaries. Consequently, all remaining augmented side components must be planar, as in Case 1. It follows that $H + ab$ is the only possible non-planar augmented side component.

Case 4. TK_5 is embedded as E_5 of Figure 9.1.

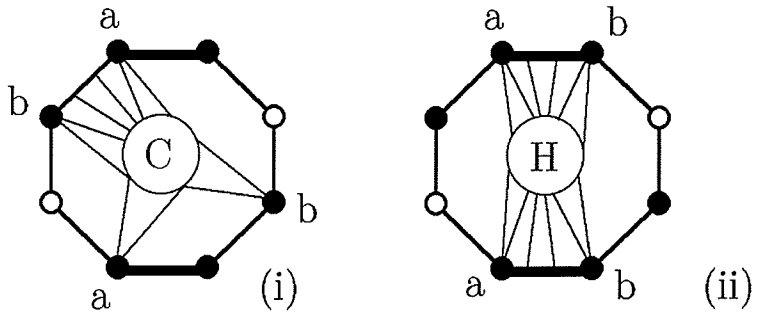

Figure 9.4: A face with three repeated vertices

The boundary of the face F of E_5 has an edge and another vertex repeated twice as in Figure 9.4. If just one corner of the side $\{a, b\}$ is repeated twice on the boundary of F, then it is equivalent to Case 2 and $H+ab$ is planar. If both corners a and b are repeated twice on the boundary of F, but the side itself appears just once (Figure 9.4(i)), we have a case similar to Case 3 and $H + ab$ is the only possible non-planar augmented side component in G. Suppose the entire side $\{a, b\}$ appears twice on the boundary of F, and H is embedded in F as in Figure 9.4(ii). Then we find that after embedding H, any face of the embedding has at most one repeated corner as in case 2. Consequently, there can be at most one non-planar augmented side component $H + ab$.

Case 5. TK_5 is embedded as E_2 of Figure 9.1.

The boundary of the face F of E_2 has two edges repeated twice as in Figure 9.5. If a and b are endpoints of a repeated edge as in Figure 9.5(ii), it is equivalent to

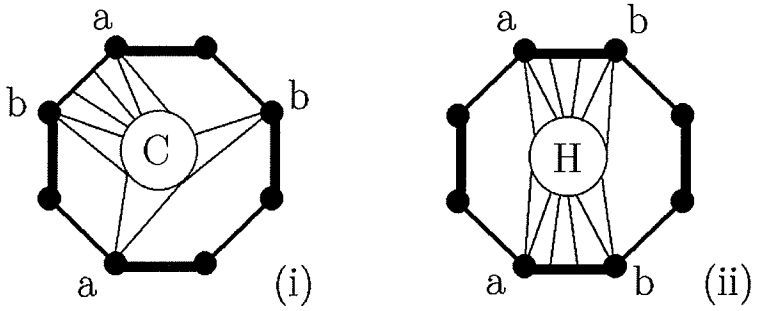

Figure 9.5: A face with four repeated vertices

Case 4 in Figure 9.4(ii). Otherwise we get the case of Figure 9.5(i) equivalent to Case 3. In both cases, if $H + ab$ is non-planar, then all the other augmented side components are planar. ∎

Corollary 9.1 *If G is toroidal, then there can be at most one non-planar side component of TK_5.*

Proposition 9.2 *If all the side components of TK_5 in G are planar and at most one of the augmented side components is non-planar, then G is toroidal.*

Proof. If all the augmented side components are planar, then by Lemma 6.1 we can embed all the side components as planar graphs with two corners on the outer face in any of the embeddings of TK_5 on the torus.

Suppose one of the augmented side components of TK_5, say $H + ab$, is not

planar. Then it is not possible to embed the corresponding planar side component H into an open disk with both corners a and b on the boundary. However, there are two embeddings of TK_5 on the torus (E_2 and E_5 of Figure 9.1) with a side appearing exactly twice on the boundary of a face. Denote such a face by F. The face F is indicated in the diagram of E_2 and E_5 of Figure 9.1, and a side which appears twice is drawn in bold. A face F with a side appearing twice on its boundary defines a cylinder. We can create a cylindrical embedding of the planar graph H as follows. Embed H on the sphere, and cut a small open disk which touches a from the interior of a face having a on its boundary, and cut another open disk which touches b from the interior of a face having b on its boundary. This converts the sphere into a finite cylinder. Now H contains an ab-path, namely the side $\{a, b\}$ of TK_5. Cut the cylinder along this ab-path to convert it into an open disk with a repeated ab-path on its boundary. This cylindrical embedding of H can then be placed in the face F of the TK_5 embedding E_2 or E_5 of Figure 9.1. Any planar rotation system of H provides such a cylindrical embedding of the side component H, and vice versa.

Since all the other augmented side components of TK_5 are planar, by Lemma 6.1 any side component different from H can be embedded in an open disk with both corners on the outer face. ■

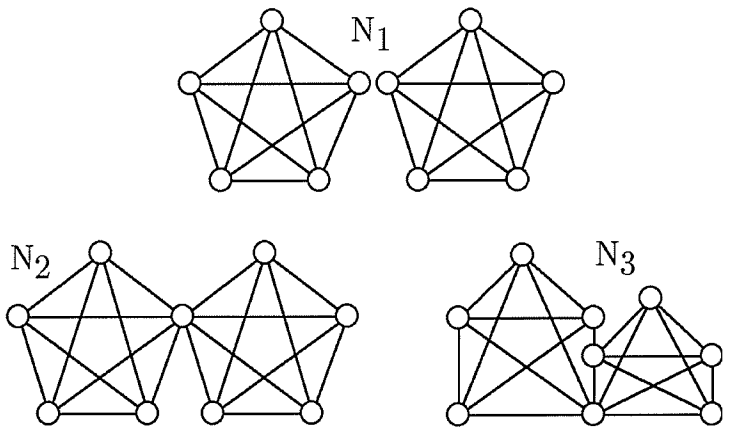

Figure 9.6: Non-toroidal graphs N_1, N_2, N_3

9.2 A Unique Non-Planar Side Component

We now consider graphs G with a non-planar side component of TK_5. Before we give an equivalent of Theorem 7.1 for the torus, we show two families of graphs which can be considered as combinations of two K_5-subdivisions having at most one side in common. One family is presented in Figure 9.6 and another in Figure 9.7.

Lemma 9.1 *None of graphs N_1, N_2 or N_3 of Figure 9.6 can be embedded on the torus.*

Proof. Consider all embeddings of K_5 on the torus (see Figure 9.1). Clearly, it is not possible to embed a non-planar component K_5 into an open disk.

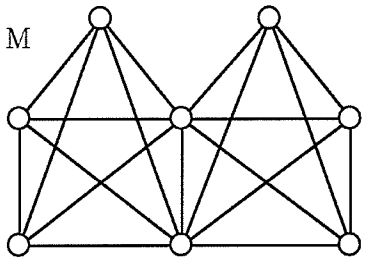

Figure 9.7: Toroidal graph M

Therefore we can not complete any of the embeddings of Figure 9.1 to graph N_1. Now to extend one of the embeddings of Figure 9.1 to N_2, it is necessary to embed K_4 into a face equivalent to an open disk and then add edges between all vertices of K_4 and one corner on the boundary of the face. Since K_4 is not outer-planar, it is not possible to do so without edge crossings. Therefore N_2 is non-toroidal. Finally, to extend one of the embeddings of Figure 9.1 to N_3, we need to embed K_4 into a face equivalent to an open disk and then add edges between all four vertices of K_4 and two corners on the boundary of the face. Since K_4 is not outer-planar, it can not be done without edge crossings. Therefore N_1, N_2 and N_3 are not toroidal. ■

It can be seen that we can complete some of the embeddings of K_5 on the torus to an embedding of graph M of Figure 9.7. We must add a K_3 into one of the faces of an embedding of K_5, and join each vertex of K_3 to two corners of K_5. This can be done in several ways (using E_2, E_4 or E_5 of Figure 9.1). Notice that any embedding of M on the torus has similar properties to the

embeddings of K_5 on the projective plane. We state them in the following lemma.

Lemma 9.2 *In any embedding of the graph M on the torus, every vertex of M appears at most once on the boundary of any face of the embedding and every edge of M is on the boundary of exactly two faces.*

Proof. A proof of Lemma 9.2 is done by adding a triangle into a face of the torus embeddings of K_5 (E_2, E_4 and E_5 of Figure 9.1) and all edges between the triangle and two corners of K_5 in all possible ways. This cuts the face F of the embeddings E_2, E_4 and E_5 of Figure 9.1 into smaller faces whose boundary has no multiple appearance of vertices and edges. ■

The graph M can be viewed as two K_5's with one edge identified. Let TM be a subdivision of M. TM contains two K_5-subdivisions, TK_5' and TK_5'', with one side in common. A *corner* of TM is a corner of any of the two TK_5's. A *side* of TM is a side of any of the TK_5's. A vertex of TM is called *inner* if it is an inner vertex in any of the TK_5's.

Now suppose graph G has TM as a subgraph and there is no short cut or 3-corner vertex of any of two corresponding TK_5's of TM in G. Then TK_5' is contained in a side component of TK_5'' in G and vice versa. As in Proposition 6.2, let K be the set of corners of TM. We define a *side component* of TM

as a subgraph in G induced by a pair of corners a and b of TK_5' or TK_5'' in TM and all connected components of $G\backslash K$ adjacent to a and b. Clearly, any two side components of TM can intersect just in the common corner of TM if one exists. An *augmented side component* of TM is defined as before. Clearly, Lemma 6.1 holds as well for the side components of TM.

The next proposition provides an alternative proof of Corollary 9.1.

Proposition 9.3 *If there is more than one non-planar side component of TK_5 in G, then G is not toroidal.*

Proof. Two side components each containing a subdivision of K_5 or $K_{3,3}$ can intersect in at most one vertex. Therefore G contains as a minor one of N_1, N_2 of Figure 9.6 or one of the graphs of Figure 9.8. This covers all possible combinations of K_5 and $K_{3,3}$.

Similar reasoning to Lemma 9.1 shows that graphs N_4, N_5, N_6 and N_7 of Figure 9.8 are not toroidal. It is not possible to embed $K_{3,3}$ into an open disk. This rules out N_4 and N_6. Consider N_5 and N_7 as $K_{3,3}$ or K_5, respectively, with one vertex adjacent to three independent vertices of $K_{2,3}$. $K_{2,3}$ is not outer-planar, yet must be embedded in an open disk. There is always one vertex of a set of three independent vertices of $K_{2,3}$ that cannot be joined to a vertex on the boundary of the disk. Consequently, it is impossible to complete an embedding of $K_{3,3}$ to N_5 or K_5 to N_7. Thus G has a non-toroidal minor. ■

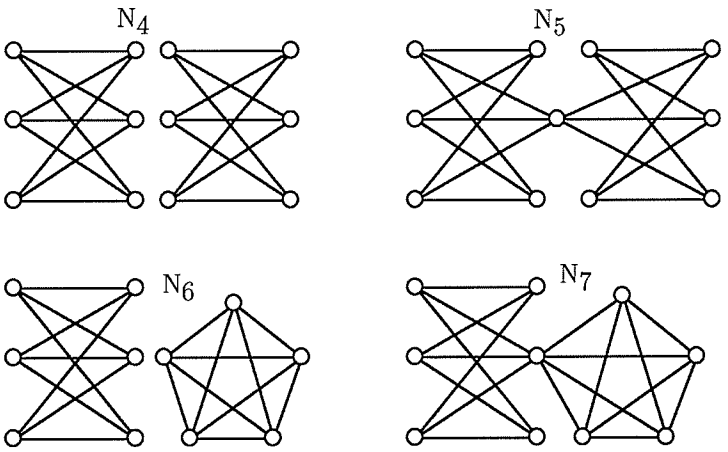

Figure 9.8: Non-toroidal graphs

Now let us assume that there is exactly one non-planar side component H of TK_5 in G. Denote the corresponding side of TK_5 by h and its corners by a and b. Suppose that H contains a K_5-subdivision TK_5' and that there is no short cut or 3-corner vertex of TK_5' in G.

Theorem 9.1 *Let graph G have a K_5-subdivision TK_5 with no short cut or 3-corner vertex in G. Let there be one non-planar side component H of TK_5 which contains a K_5-subdivision TK_5' with no short cut or 3-corner vertex in G. Then G is toroidal if and only if TK_5 and TK_5' have two common corners, and $TK_5 \cup TK_5'$ contains an M-subdivision TM all of whose augmented side components in G are planar.*

Proof. First we prove the sufficient conditions. In any embedding of TM on the torus, for each side of TM construct a planar embedding of its side component with both corners on the outer face. By Lemma 6.1, there exists such an embedding of a side component if and only if the augmented side component is a planar graph. Clearly, we can embed every side component independently to obtain an embedding of G.

Now we prove the necessary conditions. We consider all possible cases of intersection of TK_5 and TK_5' in G. If $TK_5 \cap TK_5' = \emptyset$, then G has minor N_1 of Figure 9.6 and G is not toroidal.

If $TK_5 \cap TK_5' \neq \emptyset$, let h be the side of TK_5 which is contained in a non-planar side component H, and let a and b be the corners of H. Denote by x the vertex of $h \cap TK_5'$ closest to a on side h and by y the vertex of $h \cap TK_5'$ closest to b on side h. If $x = y$, then G has minor N_2 of Figure 9.6, obtained by possibly contracting the edges of a path, and so G is not toroidal. So, $x \neq y$.

Without loss of generality, suppose $x \neq a$. The following cases are possible.

1) x is an inner vertex on a side of TK_5'.

If y is on the same side of TK_5' as x, then G contains minor N_3 of Figure 9.6 and G is not toroidal. Otherwise y is on a different side of TK_5', and TK_5 contains a short cut of TK_5' in G with endpoints x and y – a contradiction since G has no short cut of TK_5 or TK_5'.

2) x is a corner of TK_5'.

If y is on the same side of TK_5' as x, then G contains a minor N_3 of Figure 9.6 and G is not toroidal. Otherwise y is on a different side of TK_5'. Then y is an inner vertex of TK_5' and TK_5 contains a short cut of TK_5' in G with endpoints x and y – a contradiction.

Hence $x = a$ and $y = b$. Now suppose x or y is an inner vertex of TK_5'. If x and y are on the same side of TK_5', we have a minor N_3 of Figure 9.6 in G and G is not toroidal. If x and y are on different sides of TK_5', then TK_5 contains a short cut of TK_5' in G – a contradiction. Thus x and y are both corners of TK_5'.

Without loss of generality we can substitute the side h of TK_5 by the side between x and y in TK_5'. Clearly, the substitution does not create any short cut or 3-corner vertex of TK_5 in G and it does not affect the side components of TK_5 in G. On the other hand, TK_5 and TK_5' now have a common side and give us an M-subdivision TM in G. Clearly, Lemma 6.1 holds for the side components of TM in G too. By using Lemma 9.2, the same reasoning as in Theorem 7.1 shows that an embedding of TM on the torus can be extended to G if and only if all the side components of TM in G are planar with both corners on the outer face. A non-planar augmented side component of TM in G can not be added into any of the embeddings of TM on the torus. ■

9.3 Description of the Algorithm

Propositions 9.1 and 9.3 as well as Theorem 9.1 give us a linear time practical algorithm for graphs with a K_5-subdivision.

Algorithm 9.1 *Torus Embedding Algorithm for Graphs with a K_5-Subdivision.*

Input: *A 2-connected graph G*

Output: *Either a toroidal rotation system of G, or a $K_{3,3}$-subdivision in G, or an indication that G is not toroidal*

(1) Use a planarity checking algorithm (eg. [19]) to determine if G is planar. If G is planar then return its planar rotation system. If G is not planar and the planarity check returns a $K_{3,3}$-subdivision in G then return the $K_{3,3}$-subdivision in G.

(2) If G is not planar and the planarity check returned a K_5-subdivision TK_5 in G, then do a depth-first or breadth-first search to find either a short cut or a 3-corner vertex of TK_5 in G. If a short cut or 3-corner vertex is found, then return a $K_{3,3}$-subdivision in G. If there is no short cut or 3-corner vertex, the depth-first or breadth-first search returns the side components of TK_5.

(3) If there are two non-planar augmented side components of TK_5 in G, then return G is not toroidal. If there is at most one non-planar augmented

side component of TK_5 and the corresponding side components of TK_5 in G is planar, then return a toroidal rotation system of G. If the side component corresponding to the non-planar augmented side component is not planar then go to the next step.

(4) There is exactly one non-planar side component of TK_5 in G. If the planarity check for the side component returned a $K_{3,3}$-subdivision, then return the $K_{3,3}$-subdivision in G. If the planarity check for the side component returned a K_5-subdivision TK_5', then do a depth-first or breadth-first search to check if there is a short cut or a 3-corner vertex of TK_5' in G. If a short cut or a 3-corner vertex of TK_5' is found, then return a $K_{3,3}$-subdivision in G.

(5) Check if TK_5 and TK_5' have two common corners. If they do not have two common corners, then return G is not toroidal. If they do have two common corners, then construct an M-subdivision TM in G. Find the side components of TM using a depth-first or breadth-first search.

(6) For each augmented side component of TM in G, check if it is planar. If all the augmented side components are planar, then return a toroidal rotation system of G. If there is a non-planar augmented side component of TM, then return G is not toroidal.

Each step in this algorithm consists of a constant number of linear time planarity checks or of a linear time depth-first or breadth-first search. The steps

are executed in a consecutive order . Therefore the entire algorithm has a linear time complexity. The linear time planarity checks can return a planar embedding. This would provide a toroidal embedding of the whole graph.

Chapter 10

Conclusions and Future Work

The thesis describes and develops fundamental ideas of graph embedding algorithms for the plane, projective plane and torus which can be implemented by a computer program. The polygonal surface representation is used for the graph embedding problem. The polygonal surface representation allows to have planar pictures of complex graphs embedded on a surface.

In general, the graph embedding problem is difficult and hard to deal with. It is NP-complete to determine the genus of a graph (see [36]). The problem has a double nature: we need to assign a combinatorial object which is a graph to a surface that has mostly continuous properties. However it is possible to solve the embedding problems using signed rotation systems of a graph, and

by using Euler's formula. These are strictly combinatorial methods. A signed rotation system is a combinatorial representation of a graph embedding which can be constructed by a computer program.

The Hopcroft-Tarjan linear time planarity algorithm is complicated and involved with many details. However it became popular and had many improvements and modifications as in [23], [39], [40]. The projective plane and torus are the topological surfaces closest to the plane. The work presented in this thesis generalizes the ideas of the Hopcroft-Tarjan planarity algorithm and uses planarity algorithms to devise linear time algorithms for embedding graphs in the projective plane and torus.

In Chapter 4 we described new methods to transform a planar embedding of a graph into a 2-cell embedding on the projective plane and torus. It provides easy algorithms to do the transformation in $O(1)$ time and to draw planar graphs on the projective plane and torus as 2-cell embeddings.

We simplified the algorithms of Chapter 5 by using two different approaches. Each approach was based on the structural properties and embeddings of two minimal non-planar Kuratowski graphs K_5 and $K_{3,3}$ on the projective plane and torus.

The first approach is based on the structural results for K_5-subdivisions in a graph described in Chapter 6. This allows projective planarity and toroidality algorithms to be reduced to a constant number of planarity checks as in

the Hopcroft-Tarjan algorithm, or to a subdivision of the other Kuratowski graph $K_{3,3}$ in the graph. The algorithms to embed graphs containing a K_5-subdivision are presented in Section 7.1 for the projective plane, and in Chapter 9 for the torus. The algorithms are relatively easy to implement and they simplify algorithms in [21], [28] and [30]. By using the algorithm for the projective plane, we exclude 27 initial labelled embeddings of K_5 and need to consider only the remaining 6 labelled embeddings of $K_{3,3}$ on the projective plane in [28]. The algorithm of Section 7.1 was implemented by W. Myrvold (personal communication). By using the algorithm for the torus of Chapter 9, we exclude 6 unlabelled embeddings of K_5 and need to consider just 2 unlabelled embeddings of $K_{3,3}$ on the torus for [21].

We hope to develop these ideas and techniques for K_5-subdivisions to devise practical and more efficient general algorithms. Also, the approach of excluding K_5-subdivisions can likely be generalized for graph embedding algorithms in orientable and non-orientable surfaces of higher genus.

The second approach consists in considering the embedding of the other Kuratowski graph $K_{3,3}$ on the projective plane and in generalizing the Hopcroft-Tarjan algorithm. The structural results for $K_{3,3}$ embeddings on the projective plane are given in Sections 7.2 and 7.3. They result in the algorithms of Chapter 8 which yields a linear time projective planarity algorithm. The basic strategy is a development of the Hopcroft-Tarjan algorithm for the embedding of $K_{3,3}$ on the projective plane by using the special features of the embedding.

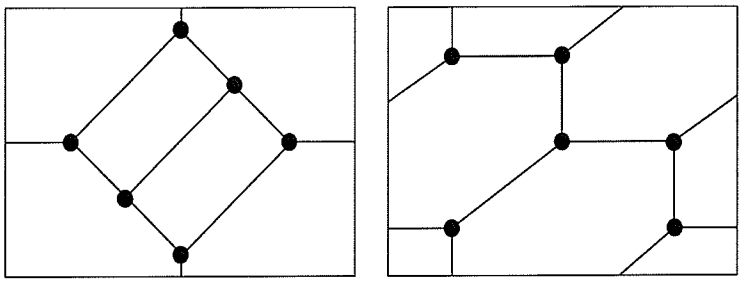

Figure 10.1: The two embeddings of $K_{3,3}$ on the torus

Future work could consist in simplifying the algorithm of Chapter 8 and in devising a practical polynomial time algorithm for toroidal graphs with a $K_{3,3}$-subdivision. The two embeddings of $K_{3,3}$ on the torus are depicted in Fig. 10.1. It is likely that the embedding of Fig. 10.1(b) can be treated in a similar way to the embedding of $K_{3,3}$ on the projective plane. However the embedding of Fig. 10.1(a) has a face with two edges of $K_{3,3}$ appearing twice on its boundary. The multiple appearance of vertices and edges on a face boundary complicates the completion of an embedding of $TK_{3,3}$. The next step of the research would be to find an efficient approach in this case as well.

We hope that ideas for the projective plane and torus will lead to an efficient and practical way to decide on an embedding of a graph into an arbitrary surface. One motivation for this problem is in its possible use for VLSI design.

Another closely related interesting problem is to distinguish different embeddings on a surface and to enumerate them. Whitney's theorem [38] says that

3-connected graphs can have at most one planar embedding up to isomorphism. As the results in [15] show, there may be many 2-cell embeddings of 3-connected graphs on the torus. We would like to use our structural and algorithmic results to see if there is an analogue of Whitney's theorem for other surfaces.

In general, we would also like to obtain all possible embeddings of a graph on a surface and determine how many different embeddings exist. As described in [15], we can make a distinction between orientable and non-orientable embeddings on an orientable surface. A regular embedding gives us a tiling of the plane. We are interested in possible classifications of the embeddings and their characterizations. Also it can be interesting to consider the dual graphs of the embeddings in more detail.

Bibliography

[1] D. Archdeacon, "A Kuratowski theorem for the projective plane", J. Graph Theory, 5 (1981), no. 3, 243-246.

[2] D. Archdeacon and J.P. Huneke, "A Kuratowski theorem for non-orientable surfaces", J. Comb. Theory Ser. B, 46 (1989), 173-231.

[3] R. Bodendiek and K. Wagner, "Solution to König's graph embedding problem", Math. Nachr., 140 (1989), 251-272.

[4] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, Elsevier Publishing, New York, 1976.

[5] J. Boyer and W. Myrvold, "Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm", Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 140-146, 1999.

[6] J. Chambers, "Hunting for Torus Obstructions", Master's thesis, University of Victoria, 2002.

[7] S.A. Cook, "The complexity of theorem-proving procedures", in *3rd ACM Symp. Theory of Comp.*, 151-158, 1971.

[8] R. Diestel, *Graph Theory*, 2nd edition, Springer, New York, 2000.

[9] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems", SIAM J. of Computing, 5 (1976), 691-703.

[10] M. Fellows and P. Kaschube, "Searching for $K_{3,3}$ in linear time", Linear and Multilinear Algebra, 29 (1991), 279-290.

[11] J.R. Fiedler, J.P. Huneke, R.B. Richter, and N. Robertson, "Computing the orientable genus of projective graphs", J. Graph Theory, 20 (1995), 297-308.

[12] I.S. Filotti, "An algorithm for embedding cubic graphs in the torus", J. Comp. Sys. Sci., 2 (1980), 255-276.

[13] M. Fréchet and K. Fan, *Initiation to Combinatorial Topology*, Prindle, Weber and Schmidt, Boston, 1967.

[14] A. Gagarin and W. Kocay, "Embedding graphs containing K_5-subdivisions", Ars Combinatoria, 64 (2002), 33-49.

[15] A. Gagarin, W. Kocay, and D. Neilson, "Embeddings of small graphs on the torus", Cubo, 5 (2003), no.2, to appear.

[16] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, 1985.

[17] J.L. Gross and T.W. Tucker, *Topological Graph Theory*, Wiley, New York, 1987.

[18] M. Henle, *A Combinatorial Introduction to Topology*, Dover Publications, New York, 1994.

[19] J.E. Hopcroft and R.E. Tarjan, "Efficient planarity testing", Journal of Assoc. Comput. Mach. 21 (1974), 549-568.

[20] J.D. Horton, "A polynomial time algorithm to find the shortest cycle basis of a graph", SIAM J. Comp., 16 (1987), 358-366.

[21] M. Juvan, J. Marinček, and B. Mohar, "Embedding graphs in the torus in linear time", Integer programming and combinatorial optimization (Copenhagen, 1995), Lecture Notes in Comput. Sci., 920, Springer, Berlin, 360-363.

[22] W. Klotz, "A constructive proof of Kuratowski's theorem", Ars Combinatoria, 28 (1989), 51-54.

[23] W. Kocay, "The Hopcroft-Tarjan planarity algorithm", 1993, unpublished, http://bkocay.cs.umanitoba.ca/G&G/G&G.html

[24] W. Kocay, "Groups & Graphs, a Macintosh application for graph theory", Journal of Combinatorial Mathematics and Combinatorial Computing 3 (1988), 195-206.

[25] K. Kuratowski, "Sur le problème des courbes gauches en topologie", Fund. Math. 15 (1930), 271-283.

[26] B. Mohar and C. Thomassen, *Graphs on Surfaces*, The Johns Hopkins University Press, Baltimore and London, 2001.

[27] O. Melnikov, R. Tyshkevich, V. Yemelichev, and V. Sarvanov, *Lectures on Graph Theory*, Bibliographisches Institut, Mannheim, 1994.

[28] B. Mohar, "Projective planarity in linear time", Journal of Algorithms 15 (1993), 482-502.

[29] B. Mohar, "A linear time algorithm for embedding graphs in an arbitrary surface", SIAM J. Discrete Math. 12 (1999), no. 1, 6–26 (electronic).

[30] W. Myrvold and J. Roth, "Simpler projective plane embedding", Ars Combinatoria, to appear.

[31] W. Myrvold and E. Neufeld, "Practical toroidality testing", Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (New Orleans, LA, 1997), 574–580, ACM, New York, 1997.

[32] B. Perunicic and Z. Duric, "An efficient algorithm for embedding graphs in the projective plane", In *The 5th quadrennial international conference on*

the theory and applications of graphs with special emphasis on algorithms and computer science applications, Western Michigan University, June 4-8, 1984, John Wiley and Sons, Inc.

[33] E.M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

[34] B. Richter and H. Shank, "The cycle space of an embedded graph", J. Graph Theory, 8 (1984), 365-369.

[35] N. Robertson and P. Seymour, "Graph minors. VIII. A Kuratowski theorem for general surfaces", J. Comb. Theory Ser. B, 48 (1990), 255-288.

[36] C. Thomassen, "The graph genus problem is NP-complete", J. Algorithms, 10 (1989), 568-576.

[37] C. Thomassen, "The Jordan-Schönflies Theorem and the Classification of Surfaces", Amer. Math. Monthly, 99 (1992), no. 2, 116-131.

[38] H. Whitney, "2-isomorphic graphs", American J. of Math., 55 (1933), 245-254.

[39] S.G. Williamson, "Embedding graphs in the plane – algorithmic aspects", Ann. Disc. Math., 6 (1980), 349-384.

[40] S.G. Williamson, "Depth-first search and Kuratowski subgraphs", Journal of Assoc. Comput. Mach., 31 (1984), 681-693.

[41] S.G. Williamson, Math. Reviews, 94f:05141, 1994.