A Faster Algorithm for Torus Embedding

by

Jennifer Roselynn Woodcock
B.Sc. University of Victoria 2004

A Thesis Submitted in Partial Fullfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

A Faster Algorithm for Torus Embedding

by

Jennifer Roselynn Woodcock
B.Sc. University of Victoria 2004

Supervisory Committee

Dr. Wendy Myrvold (Department of Computer Science)
Supervisor

Dr. Frank Ruskey (Department of Computer Science)
Co-Supervisor

Dr. Ulrike Stege (Department of Computer Science)
Departmental Member

Dr. Luis Goddyn (Department of Mathematics, Simon Fraser University)
Outside Member

Supervisory Committee

Dr. Wendy Myrvold (Department of Computer Science)
Supervisor

Dr. Frank Ruskey (Department of Computer Science)
Co-Supervisor

Dr. Ulrike Stege (Department of Computer Science)
Departmental Member

Dr. Luis Goddyn (Department of Mathematics, Simon Fraser University)
Outside Member

Abstract

Although theoretically practical algorithms for torus embedding exist, they have not yet been successfully implemented and their complexity may be prohibitive to their practicality. We describe a simple exponential algorithm for embedding graphs on the torus (a surface shaped like a doughnut) and discuss how it was inspired by the quadratic time planar embedding algorithm of Demoucron, Malgrange and Pertuiset. We show that it is faster in practice than the only fully implemented alternative (also exponential) and explain how both the algorithm itself and the knowledge gained during its development might be used to solve the well-studied problem of finding the complete set of torus obstructions.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

This research was supported financially partly by a Fellowship from the University of Victoria and partly by a Canada Graduate Scholarship from NSERC.

I am grateful to my supervisors, Wendy Myrvold - whose helpful ideas and patience in discussions have made this thesis possible - and Frank Ruskey. They provided me with the constant encouragement and support I needed to believe I could do this.

Thanks are also due to my family and friends who think it is fabulous that I do research about drawing pictures on doughnuts, and who encouraged me through the troubles of life while I worked on this thesis. Finally, I must thank Joe Whitney and Mark Weston because without their input and caring support I don't think I would have started this journey to begin with.

Chapter 1

Introduction

Graph theory can be used to model many tangible or abstract problems that involve connections between objects. A sociologist might, for example, use a graph to model patterns of relationships in a population; in electronic engineering, a graph might be used to model electronic circuitry before it is physically laid out; and in chemistry, graphs have been used to model carbon molecules known as fullerenes (e.g. [18, 17]). Whatever the application, it is frequently visually appealing and useful to see a drawing of the graph model that has few or, preferably, no edges that cross.

Graphs that can be drawn, or embedded, on the plane with no crossing edges are not only visually appealing: there are graph theory problems which are hard in the general case but have efficient solutions for planar graphs (see [21], page 285, for example). Further, interest in graph embedding is not limited to the plane. There are infinitely many surfaces for which we can find the "embeddable" graphs and knowledge of these graphs has the potential to further research in the field. In this thesis, we focus on the surface known as the torus, which is shaped like a doughnut. As with the plane, there are graph theory problems which are hard in the general case but have efficient solutions for toroidal graphs (see [20, 37] for example).

Several efficient and practical algorithms for embedding graphs on the plane (e.g. [24, 15]) and at least one for the projective plane [36] have been implemented. For the torus, however, although theoretically efficient algorithms exist [26, 27], they are complex, meaning that programming them can be error-prone and that proving the correctness of either the theory or an implementation is a challenge. It is also probable that they have a high constant in their running times, making them less efficient for small graphs. As of yet, only exponential algorithms which have practical limitations have been completely and correctly implemented [34, 33]. We introduce a new exponential algorithm for embedding graphs on the torus that has been implemented and tested, and is conceptually simpler as well as faster in practice than previous algorithms. It is feasible that this algorithm and the insights gained during its development will play a role in finally concluding the search for all of the torus obstructions - a well-studied yet still open problem - which would be a major breakthrough in topological graph theory.

1.1 Overview of Thesis

In Chapter 2 we provide definitions and concepts necessary for understanding the material and we present a more in-depth history of the relevant and related research that motivated this work. We introduce a generic embedding framework for embedding graphs on surfaces in Chapter 3. The new torus embedding algorithm that is the main subject of this thesis was inspired by the quadratic algorithm of Demoucron, Malgrange, and Pertuiset for embedding graphs on the plane [15]. Also in Chapter 3, therefore, as a preface to the presentation of our algorithm, we explain how their algorithm fits into the generic framework.

In Chapter 4 we discuss the details of our new torus embedding algorithm, why it is more complex than the algorithm of Demoucron, Malgrange, and Pertuiset, why it has exponential instead of quadratic running time, and why it is correct. Chapter 5 presents timing comparisons between our new algorithm and a previous exponential algorithm that show significant improvements. Finally, Chapter 6 concludes this thesis by outlining possible approaches to enhancing our algorithm and using it to find all of the torus obstructions.

Chapter 2

Background

Elementary concepts and definitions in graph theory and their proofs can be found in introductory texts by West [39] and Bondy and Murty [9]. Archdeacon's survey of topological graph theory gives an excellent introduction to the study of graph embeddings [3]. More in-depth discussions can be found in Henle's book *A Combinatorial Introduction to Topology* [23] and in Mohar's and Thomassen's book *Graphs on Surfaces* [31]. Here we present basic definitions and key concepts pertinent to understanding the material presented in this thesis followed by a history of related and relevant work that provided motivation for this research.

2.1 Basic Graph Theory Definitions

A *graph* $G = \{V, E\}$ consists of a finite set V of *vertices* and a finite set E of *edges* where each edge $e = (u, v)$ of E is associated with an unordered pair of vertices u and v from V. Vertices u and v are the *endpoints* of edge $e = (u, v)$ and edge e is *incident* to a vertex u if and only if u is an endpoint of e. The *neighbours* of a vertex u are the vertices v such that $(u, v) \in E$. The *degree* of a vertex u in a graph is the number of edges that are incident to u. The *minimum degree* of a graph G is the minimum among the degrees of all of the vertices of G. An *r-regular* graph has only vertices with degree r. For this thesis, we consider only *simple graphs* which do not have *multiple edges* (more than one edge with the same pair of endpoints) or *loops* (edges of the form (u, u)). For conciseness throughout this thesis we let n denote the number of vertices, or the *order*, of a graph and m denote the number of edges, or the *size*, of a graph. When discussing running times we do so in terms of n, the number of vertices. Thus $O(n)$ is linear time, $O(n^2)$ is quadratic time, and so on.

To *delete a vertex* v from a graph G (denoted $G - v$) is to remove v and all edges incident to v from G. To *delete an edge* e from a graph G (denoted $G - e$) is to remove e from G. To *contract* an edge $e = (u, v)$ in a graph G (denoted $G \cdot e$) is to remove e from G, replace vertices u and v by a single vertex w, and replace any edges (u, x) incident to u and (v, y) incident to v by (w, x) and (w, y) respectively. Contracting an edge can create multiple copies of an edge (w, x) if u and v were both adjacent to some vertex x. Since we are only interested in simple graphs any multiple edges are removed when an edge is contracted.

A *walk* in a graph $G = \{V, E\}$ is a finite alternating sequence of vertices and edges of the form

$$v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, v_{k-2}, (v_{k-2}, v_{k-1}), v_{k-1}$$

where $v_0, v_1, \ldots, v_{k-1} \in V$ and $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-2}, v_{k-1}) \in E$; we say that such a walk is between vertices v_0 and v_{k-1}, and v_0 and v_{k-1} are the *endpoints* of the walk. A *closed walk* is a walk in which $v_0 = v_{k-1}$. A *path* in G is a walk with no repeated vertices and a *cycle* in G is a closed walk with no repeated vertices except $v_0 = v_{k-1}$. A graph G *is* a path of length n if it consists of n distinct vertices $v_0, v_1, \ldots, v_{n-1}$ and the edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-2}, v_{n-1})$. A graph G *is* a cycle of length n if it consists of n distinct vertices $v_0, v_1, \ldots, v_{n-1}$ and the edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-2}, v_{n-1}), (v_{n-1}, v_0)$.

Subdividing an edge (u, v) of a graph means replacing (u, v) by a vertex w and two edges (u, w) and (w, v). In a graph, *suppressing* a path

$$v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, v_{k-2}, (v_{k-2}, v_{k-1}), v_{k-1}$$

which has the property that the degrees of $v_1 \ldots v_{k-2}$ are all equal to two, means replacing

$$(v_0, v_1), v_1, (v_1, v_2), \ldots, v_{k-2}, (v_{k-2}, v_{k-1})$$

by the edge (v_0, v_{k-1}). A graph H is *homeomorphic* to a graph G if H can be obtained from G by a sequence of edge subdivisions and path suppressions; graph H is a *homeomorph* of graph G. A graph $H = \{V', E'\}$ is a *subgraph* of a graph $G = \{V, E\}$ if $V' \subseteq V$ and $E' \subseteq E$.

A *complete graph* on n vertices, K_n, is a graph $G = \{V, E\}$ with $|V| = n$ and $E = \{(u, v)|u, v \in V \textit{ and } u \neq v\}$. A *bipartite graph* $G = \{V \cup V', E\}$ is a graph whose vertices can be partitioned into two sets V and V' such that there are no edges between vertices in the same set. A *complete bipartite graph* on $x + y$ vertices, $K_{x,y}$, is a bipartite graph $G = \{V \cup V', E\}$ such that $|V| = x$ and $|V'| = y$ and $E = \{(u, v)|u \in V \textit{ and } v \in V'\}$. This thesis often refers to a graph containing a subgraph H homeomorphic to K_5 or $K_{3,3}$. The vertices of H that have degree greater than or equal to three are referred to as *main vertices*.

Figure 2.1 shows K_5 on the left-hand side and a graph homemorphic to K_5 on the right-hand side; the main vertices of the K_5 are coloured grey to differentiate them from the other vertices. Figure 2.2 shows $K_{3,3}$ on the left-hand side and a graph homeomorphic to $K_{3,3}$ on the right-hand side; the main vertices of the two sets are coloured with two shades of grey to differentiate them from one another and from the other vertices.

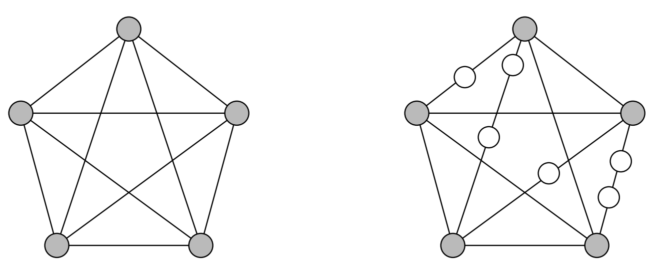


Figure 2.1: K_5 and a graph homeomorphic to K_5.

A graph G is *connected* if there is a path between every pair of vertices in G. A *connected component* of a graph G is a maximal connected subgraph of G. A *cut-*

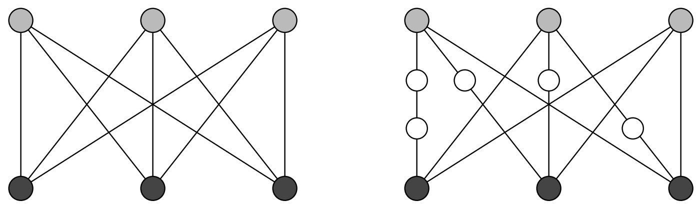


Figure 2.2: $K_{3,3}$ and a graph homeomorphic to $K_{3,3}$.

vertex in a graph G is a vertex v of G such that if we delete v from G the resulting graph is not connected. A graph that is connected and does not have any cut-vertices is called *2-connected*. A *block* of a graph G is a maximal 2-connected subgraph of G. A *2-vertex cut*, $\{a, b\}$, in a graph G is a pair of vertices a and b from G such that if we delete vertices a and b from G the resulting graph is not connected.

A *bridge* B with respect to a subgraph H of a graph G is either

Type 1: a connected component C of $G - H$ along with the edges (u, v) such that $u \in C$ and $v \in H$ and the vertices $v \in H$ that are endpoints of these edges, or

Type 2: an edge (u, v) and its endpoints where $(u, v) \in G$ and $u \in H$ and $v \in H$ but $(u, v) \notin H$.

The vertices that are in both B and H are called *attachment vertices* of B. The vertices that are in B but not in H (i.e. the vertices in B that are not attachment vertices) are called *internal vertices* of B. A *bisecting path* in a bridge B is a path P that contains only vertices in B, and where $v \in P$ is an attachment vertex of B if and only if v is an endpoint of P.

2.2 Graphs and Surfaces

Topologically, a surface can be uniquely defined by its *genus (pl. genera)* (the number of *handles* in the orientable case or *crosscaps* in the nonorientable case - see

[31] for the definitions of these terms) and whether it is *orientable* or *nonorientable* (whether or not it has a well-defined sense of clockwise). The orientable surfaces are the *sphere*, which is the only surface of genus zero, and the *k-handled-torus* ($k > 0$), which has genus k. The 1-handled torus is usually referred to simply as the *torus* and can be envisioned as a doughnut-shaped surface. The nonorientable surfaces of genus one and two are called the *projective plane* (equivalent to a disk with antipodal points identified), and the *Klein bottle*, respectively. Further understanding of surfaces in a topological sense is not necessary for comprehension of this thesis.

2.2.1 Embedding Graphs on Surfaces

In a drawing of a graph, two edges *cross* if one edge passes through the other at any point (possibly an endpoint), but not if the two simply meet at an endpoint. A graph is *embeddable* on a surface S if it can be drawn on S such that no pair of edges cross. A graph is embeddable on the sphere if and only if it is embeddable on the plane, and thus graphs that are embeddable on the sphere are called *planar*; we hereafter refer to embedding graphs on the plane rather than the sphere. Graphs that are embeddable on the projective plane and torus are called *projective planar* and *toroidal* respectively.

An *embedding* of a graph on a surface S is a particular drawing, or a description of a particular drawing, of the graph on S; in other words it is a way of describing how the graph can be embedded on S. An *orientable combinatorial embedding* (sometimes called a *rotation system*) for a graph G on an orientable surface S is an adjacency list representation of G with the neighbours of each vertex ordered cyclically based on the embedding of G on S. Additional information is required for a nonorientable combinatorial embedding; we omit description of it here as it is not relevant to the subject of this thesis.

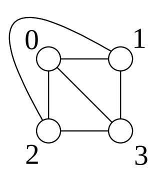
For the purpose of graph embedding, we consider two combinatorial embeddings to be equivalent if one can be transformed into the other by either

- a cyclic rotation of the adjacency list for any vertex or
- the reversal of the adjacency lists for all vertices.

Figure 2.3 depicts three combinatorial embeddings of K_4 on the plane that are equivalent. Embedding (b) differs from (a) by a cyclic rotation of the adjacency lists for vertices 0 and 3. Embedding (c) differs from (b) by a reversal of the adjacency lists for all four vertices. Notice that embeddings (a) and (b) in the figure have identical graph drawings, but (c) is a reflection of the first two drawings about edge $(0, 3)$.

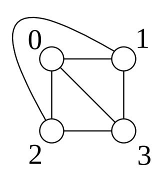
(a)

0:	1	3	2
1:	0	2	3
2:	0	3	1
3:	0	1	2



(b)

0:	3	2	1
1:	0	2	3
2:	0	3	1
3:	2	0	1



(c)

0:	1	2	3
1:	3	2	0
2:	1	3	0
3:	1	0	2

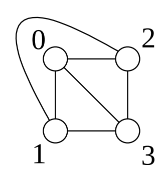


Figure 2.3: Three equivalent combinatorial embeddings of K_4 on the plane.

The genus of a combinatorial embedding can be defined combinatorially by the formula in Theorem 2.2.1, commonly known as Euler's formula.

Theorem 2.2.1. *[23] For an orientable surface S, the genus of an orientable combinatorial embedding on S of a connected graph with n vertices, m edges, and f faces is equal to*

$$\frac{2 - n + m - f}{2}.$$

□

Topologically, the *orientable (nonorientable) genus* of a graph G is the minimum of the genera of the orientable (nonorientable) surfaces on which G is embeddable. As graph theorists rather than topologists, we find it more useful to define the genus of a graph combinatorially. The *orientable (nonorientable) genus* of a graph G is the minimum of the genera of the orientable (nonorientable) combinatorial embeddings of G. For the remainder of this thesis we refer only to orientable surfaces unless otherwise specified. Thus, a combinatorial embedding will be an orientable combinatorial embedding and the genus of a graph or an embedding will be its orientable genus.

The correctness of the algorithms described in Chapters 3 and 4 hinges on the input graphs being 2-connected. Consider the following theorems about the genus of a graph.

Theorem 2.2.2. *[7, 12] The genus of a graph is equal to the sum of the genera of its blocks.* □

Theorem 2.2.3. *[7, 12] The genus of a graph is equal to the sum of the genera of its connected components.* □

Because of Theorems 2.2.2 and 2.2.3, we can use a routine which splits the input graph into its blocks as a wrapper for any code for checking the genus of a graph. Suppose we are given two algorithms, Planar(graph G) and Toroidal(graph G), which, for input graphs that are 2-connected, return *true* if the graph is planar or toroidal respectively, and *false* otherwise. Algorithm 2.1 gives pseudocode for an algorithm to find out whether or not *any* graph is toroidal or planar. For the remainder of this thesis, therefore, we avoid repetition by assuming that all input graphs to the embedding algorithms are 2-connected.

Algorithm 2.1 Components_and_Blocks(graph G)

```
Set g = 0
for all components C in G do
    for all blocks B of C do
        if Planar(B) returns true then
            Set g = g + 0
        else if Toroidal(B) returns true then
            Set g = g + 1
        else
            return G is not planar or toroidal.
        end if
    end for
end for
if g is 0 then
    return G is toroidal and planar.
else if g is 1 then
    return G is toroidal.
else
    return G is not planar or toroidal.
end if
```

2.2.2 Drawing Graphs on the Torus

Since this thesis focuses on embedding graphs on the torus, and figures will be used as an aid to understanding throughout, it is useful to explain a simple way to draw graphs embedded on the torus on paper. Figure 2.4 shows the process by which we transform the torus from a 3-dimensional doughnut shape to a 2-dimensional rectangle shape. We first cut vertically through the torus to form a cylinder and then cut horizontally through the cylinder to form a rectangle. As shown by the arrows on the arcs of the rectangle, corresponding points on the left- and right-hand arcs are identified, and corresponding points on the top and bottom arcs are identified. So when a graph is drawn on the rectangle representation of the torus, an edge which exits the rectangle through one arc, enters at the corresponding point on the opposite arc.

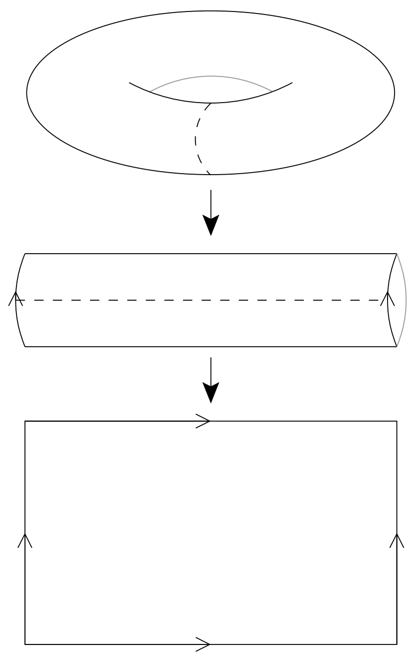


Figure 2.4: Cutting the torus to simplify drawings of graphs embedded on the torus.

Figure 2.5 shows a toroidal graph $K_{3,3}$ and one of its embeddings drawn on the rectangular representation of the torus. Note that if an edge leaves at a corner of the rectangle, it returns at the diagonally opposite corner. Henle's book *A Combinatorial Introduction to Topology* similarly describes how to draw pictures of graphs embedded on orientable and nonorientable surfaces of arbitrary genus on paper [23, pages 109 & 112].

2.2.3 Faces and How to Find Them

Given a surface S and an embedding of a graph G on S, a *face* of the embedding is a closed walk of G that bounds a maximal contiguous region of S. More formally, given a combinatorial embedding Π of graph G, let $\Pi_v(u)$ be the vertex which follows vertex u in the cyclic list of neighbours for vertex v. Then a face of Π is a minimal closed walk, $v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, (v_{k-1}, v_0)$, such that $\Pi_{v_i}(v_{(i-1) \bmod k}) = v_{(i+1) \bmod k}$

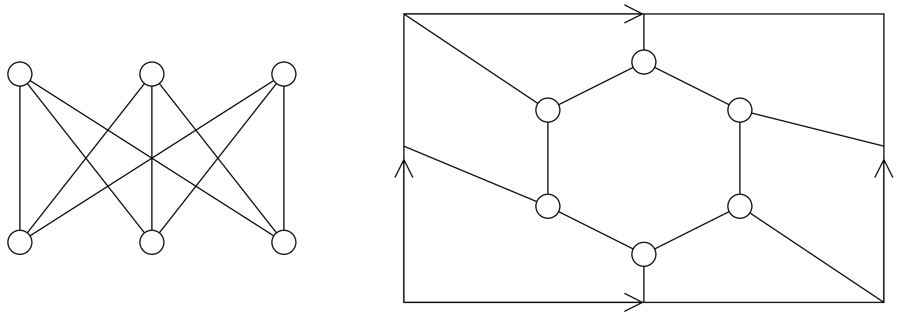


Figure 2.5: $K_{3,3}$ and one of its embeddings on the torus.

for $i = 0 \ldots k-1$. The *size* of a face is the number of vertices or edges in that face.

Figure 2.6 shows a labelled embedding of $K_{3,3}$ on the torus (a different embedding from that of Figure 2.5), and the three faces of the embedding. The ten-vertex face illustrates why we must define a face as a closed walk and not a cycle: vertices 1, 3, 4, and 6 are repeated on this face. We call faces with repeated vertices *ugly faces*.

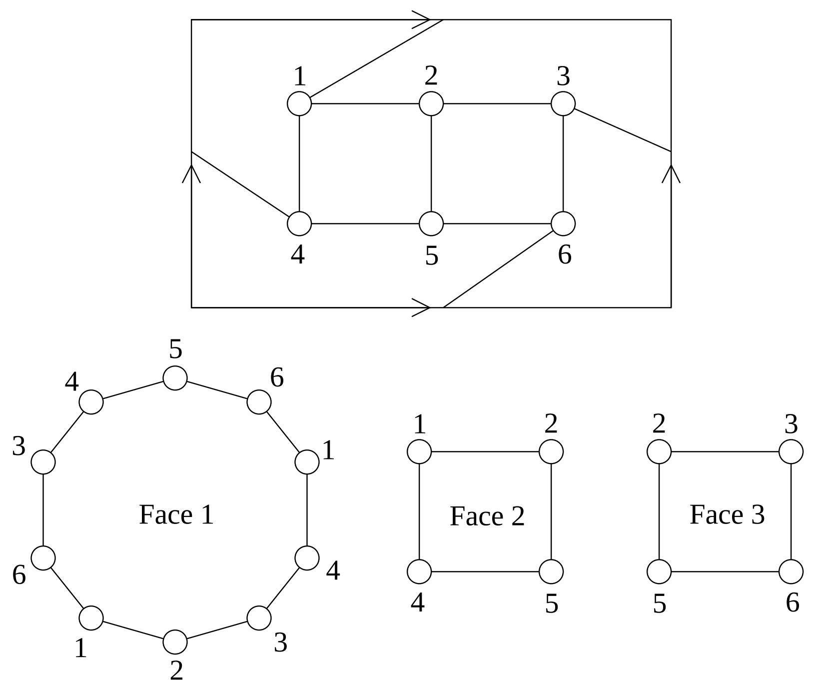


Figure 2.6: An embedding of $K_{3,3}$ on the torus and its three faces.

Suppose we have an embedding $\Pi(H)$ of a subgraph H of a graph G, that f is a face of $\Pi(H)$, and that B is a bridge of G with respect to H. Then f is *admissible* for B if all of the attachment vertices of B are in f. Also, B is *embeddable* in f if

- f is admissible for B, and
- there is a way to draw B in the contiguous region bounded by f such that no two edges cross.

Two bridges B_1 and B_2 *hinder* each other with respect to a face f if

- f is admissible for B_1 and for B_2, and
- it is not possible to draw both B_1 and B_2 in the contiguous region bounded by f such that no two edges cross.

In other words, B_1 and B_2 hinder each other with respect to f if they are not embeddable in f at the same time.

If f is an ugly face, the choice of which copies of the attachment vertices of a bridge B are used can be crucial to B being embeddable in f. For example, suppose a face f has a repeated vertex v and is admissible for a bridge B. If v is one of the attachment vertices of B, it is possible that B must connect to both copies of v in order to be embeddable in f. Figure 2.7 shows a bridge that is not embeddable in face f unless both copies of two repeated vertices of f are used (f is Face 1 of Figure 2.6).

There is a simple algorithm for finding the faces of a combinatorial embedding of a graph G. First, since each edge (u, v) should appear twice as a consecutive pair in an adjacency list (once in the order u, v, and once in the order v, u), assign two records, $[u, v]$ and $[v, u]$, to each edge (u, v) in G and initialize all records to be unvisited. Then, as long as at least one unvisited record remains, choose some unvisited record

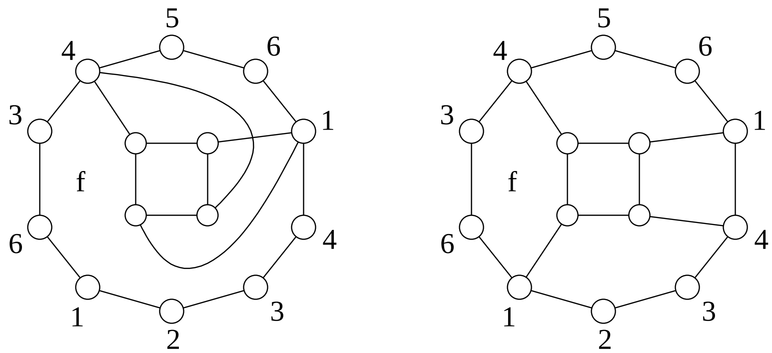


Figure 2.7: Using repeated vertices to embed a bridge in a face.

$[a, b]$ and walk around the face that begins with that record as follows. Until a visited record is reached (at which point we have finished with this face), proceed to the next record, $[b, c]$, where $c = \Pi_b(a)$, marking each record as visited along the way. Algorithm 2.2 gives pseudocode for the face walking algorithm. Myrvold and Roth provide a more in depth discussion of this algorithm [36], including how to modify it for nonorientable surfaces.

Algorithm 2.2 FaceWalk(graph G, combinatorial embedding $\Pi(G)$)

```
Let f = 0 be the number of faces seen so far.
for all edges (u, v) in G do
   Create two records [u, v] and [v, u].
end for
for all Records [a, b] do
   if [a, b] is not visited then
      f = f + 1
      while [a, b] is not visited do
         Mark [a, b] as visited.
         Add [a, b] to face f.
         Let c = Π_b(a).
         Set a = b and b = c.
      end while
   end if
end for
```

2.2.4 Obstructions

A *topological obstruction* for a surface S is a graph G with minimum degree three that is not embeddable on S but for every edge e of G, $G - e$ (G with edge e deleted) is embeddable on S. A *minor-order obstruction* for a surface S is a graph G that is a topological obstruction for S with the additional property that for every edge e of G, $G \cdot e$ (G with edge e contracted) is embeddable on S.

Suppose we are given an algorithm Torus_Embed(graph G) that answers the question "Is G toroidal?". We can easily design another algorithm Torus_Obstruction(graph G) to answer the question "Is G a torus obstruction?". First, if G is toroidal then it is obviously not a torus obstruction. Otherwise, if $G - e$ is toroidal for every edge e then G is a topological torus obstruction. Further, if $G \cdot e$ is also toroidal for every edge e then G is a minor-order torus obstruction. Algorithm 2.3 gives simple pseudocode for this algorithm, assuming that Torus_Embed(graph G) returns *true* if G is toroidal and *false* if G is not toroidal.

Algorithm 2.3 Torus_Obstruction(graph G)

```
if G has minimum degree less than three or G is toroidal then
   return  G is not a torus obstruction.
end if
for all edges e in G do
   if Torus_Embed(G − e) returns false then
      return  G is not a torus obstruction.
   end if
end for
for all edges e in G do
   if Torus_Embed(G · e) returns false then
      return  G is a topological obstruction but not a minor-order obstruction.
   end if
end for
return  G is a minor-order obstruction.
```

A torus obstruction must have at least $m = 3n/2$ edges since every vertex must have degree three or more. Euler's formula (Theorem 2.2.1) can be used to calculate the maximum number of edges a torus obstruction can have. Suppose we have a toroidal graph G with m edges and n vertices. The maximum number of faces an embedding of G can have is $f = 2m/3$ if all of the faces have size three. Substituting $f = 2m/3$ into Euler's formula and solving for m, we find that $m = 3n$ is the maximum number of edges a toroidal graph can have. Thus a torus obstruction can have at most $m = 3n + 1$ edges.

2.3 History and Motivation

Many complex linear time algorithms for embedding graphs on the plane exist, including those of Hopcroft and Tarjan (this was the first, developed in 1974) [24], Booth and Lueker [10], Fraysseix and Rosentiehl [14], Williamson [40, 41], and Boyer and Myrvold [11]. Less complex are the $O(n^2)$ algorithms of Klotz [28] and of Demoucron, Malgrange, and Pertuiset [15]. The latter of these provided the inspiration for the torus embedding algorithm presented in this thesis. For embedding graphs on the projective plane, there is a complex linear time algorithm designed by Mohar [30], and a less complex $O(n^2)$ algorithm designed and implemented by Myrvold and Roth [36] .

For the torus, there is currently no known implementation of an efficient embedding algorithm. Mohar proposed a linear time algorithm for embedding graphs on the torus [26] and Juvan and Mohar simplified the linear time algorithm to create an $O(n^3)$ variant [27]. Neither of these algorithms has yet been successfully implemented and it is possible that their complexity will be prohibitive to their practicality [32]. An exponential torus embedding algorithm was developed by Myrvold and Neufeld [34, 33] and enhanced by Chambers [12], and is practical for small graphs. Filotti also presented a specialized polynomial-time algorithm for embedding only 3-regular graphs on the torus [16], but Myrvold and Kocay proved that it is incorrect [32]. Myrvold and Kocay also discuss critical design issues in finding a polynomial time algorithm for embedding graphs on the torus [32].

In addition to the graph embedding problem in general, the research in this thesis was inspired by the problem of finding the complete set of obstructions for surfaces, and, more specifically, the problem of finding the complete set of torus obstructions. Kuratowski proved that there are there are two obstructions for the plane, K_5 and $K_{3,3}$ [29] (Wagner showed that these are also the minor-order obstructions [38]), and gave Theorem 2.3.1, now known as *Kuratowski's Theorem*.

Theorem 2.3.1. *[29, 38]* **Kuratowski's Theorem** *A graph is planar if and only if it does not contain a subgraph homeomorphic to K_5 or $K_{3,3}$.* □

That the number of obstructions is finite for any surface of fixed genus was proved by Bodendiek and Wagner for orientable surfaces [8], by Archdeacon and Huneke for nonorientable surfaces [2], and independently by Robertson and Seymour for both orientable and nonorientable surfaces [35]. This fact leads to Theorem 2.3.2, a generalization of Kuratowski's Theorem to surfaces other than the plane.

Theorem 2.3.2. *[8, 2, 35]* **Generalization of Kuratowski's Theorem** *A graph is embeddable on a surface S if and only if it does not contain a subgraph homeomorphic to one of a finite number of obstructions for S.* □

To date the only surface other than the plane for which the complete obstruction set is known is the projective plane; Glover, Huneke, and Wang listed 103 topological obstructions and 35 minor-order obstructions for the projective plane [22] and Archdeacon proved that this is a complete list [1]. Archdeacon also found the complete set of 21 3-regular topological obstructions for the spindle, the surface formed by identifying two points on the sphere [4].

Finding the complete obstruction set for the torus is a natural next step in this field of research. The exponential torus embedding algorithm of Myrvold and Neufeld is practical enough to have found all torus obstructions on up to ten vertices by exhaustive search, as well as some larger ones [34, 33]. Almost ten years later, Chambers used this algorithm to find all torus obstructions on up to eleven vertices and all 3-regular torus obstructions on up to 24 vertices by exhaustive search. He also used a "Split-Delete" approach on known small obstructions to generate a collection of larger obstructions [12]. Further, Juvan found the complete set of 270 projective planar torus obstructions [25] and Chambers, Gagarin and Myrvold found the complete set of torus obstructions which do not contain a subgraph homeomorphic to $K_{3,3}$ [13]. Although the sets of obstructions found by researchers overlap in some cases, each provided significant contributions to the knowledge base about this problem. In all, 239,451 topological obstructions have been found for the torus of which 16,683 are minor-order obstructions.

Completing the search for torus obstructions could have important theoretical and algorithmic implications. The known obstruction sets for the plane and projective plane have been used in proofs of theorems in topological graph theory. The complete set of torus obstructions could yield similar types of results. As a side-effect, it is hoped that the complete set of torus obstructions and the insights gained in finding it will provide inspiration for a faster and simpler torus embedding algorithm. Such an algorithm could be verified by ensuring that what it purports to be a minimal non-toroidal subgraph is in fact in the database of torus obstructions. Having a fast and correct torus embedding algorithm, in turn, has algorithmic implications as there are computationally intractable problems that can be solved in polynomial time for toroidal graphs (e.g. [20, 37]).

Chapter 3

Generic and Planar Embedding Algorithms

In this chapter, we first present a generic backtracking approach for embedding graphs on orientable surfaces. Then, we describe the modifications to this generic algorithm made by Demoucron, Malgrange, and Pertuiset [15] to create a linear time planar embedding algorithm which was the inspiration for our new torus embedding algorithm.

3.1 Generic Embedding Algorithm on an Orientable Surface

Algorithms 3.1 and 3.2 below present pseudocode for a generic graph embedding algorithm that can be used to decide whether any graph G of genus greater than or equal to g can be embedded on an orientable surface S of genus g. The algorithm begins by finding a subgraph H of G such that all embeddings of H on S divide S into faces that are homeomorphic to a planar disk or, informally, faces that do not have holes. One possible choice for H, when embedding a graph on a surface of genus $g > 0$, is an obstruction to a surface of genus $g - 1$.

Algorithm 3.1 StartGeneric(graph G, surface S, genus $g(S)$)

if G has genus $< g(S)$ **then**
    halt an embedding of G on a surface with genus $< g(S)$ is also an embedding of G on S.
end if
Choose a subgraph H of G that always embeds on S by dividing S into faces homeomorphic to a planar disk.
for all labelled embeddings $\Pi(H)$ of H **do**
    Generic(G, H, $\Pi(H)$)
end for

Algorithm 3.2 Generic(graph G, graph G', embedding $\Pi(G')$)

Use Algorithm 2.2 to find the faces of $\Pi(G')$.
Find the bridges of G with respect to G'.
if there are no bridges remaining **then**
  halt $\Pi(G')$ is an embedding of G.
else if there is a bridge with no admissible faces **then**
  return $\Pi(G')$ cannot lead to an embedding of G (backtrack).
end if
Choose a bridge B of G.
for all admissible faces f for B **do**
  Choose a bisecting path P of B.
  for all ways of embedding P in f **do**
    Embed P in $\Pi(G')$.
    Generic(G, G', $\Pi(G')$)
    Remove P from $\Pi(G')$.
  end for
end for

The algorithm maintains an an embedding $\Pi(G')$ of a subgraph G' of G. For each embedding $\Pi(H)$ of H, it initializes $G' = H$ and $\Pi(G') = \Pi(H)$ and then finds the faces of $\Pi(G')$ and the bridges of G with respect to G'. If there are no bridges then the algorithm must have discovered an embedding of G on S. If there is a bridge with no admissible faces, the algorithm must backtrack as $\Pi(G')$ cannot lead to an embedding of G. Otherwise, the algorithm chooses a bisecting path P from some bridge B and, for each admissible face f for B, embeds P across f in all possible ways (recall from section 2.2.3 that faces may have repeated vertices). For each way of embedding P, we recursively try to embed the rest of G.

3.2 Democron's Planar Embedding Algorithm

The planar embedding algorithm of Demoucron, Malgrange and Pertuiset [15] (which we hereafter refer to as *Demoucron's algorithm*) inspired the development of our new torus embedding algorithm. Thus, explaining how it differentiates from the generic algorithm will likely ease understanding of our new torus embedding algorithm in the following chapter.

First, if a graph does not contain any cycles then it is obviously planar. A graph that is a cycle has exactly two equivalent embeddings on the plane. So Demoucron's algorithm first finds a subgraph H of G that is a cycle and then proceeds to embed the bridges of G with respect to H as described for the generic algorithm with the following modification. It is critical to the correctness of the algorithm that bridges with only one admissible face are chosen first, but if no such bridges exist, any bridge can be selected. For conciseness, we can state that the chosen bridge must have a minimum number of admissible faces. The authors showed that as long as this is the case, it is always sufficient to choose a single admissible face for the chosen bridge in which to embed a bisecting path without trying any other admissible faces. Demoucron's algorithm is given in pseudocode in Algorithms 3.3 and 3.4 [15].

Algorithm 3.3 StartDemoucron(graph G)

if G does not contain any cycles **then**
    return G is trivially planar.
end if
Choose a subgraph H of G that is a cycle.
Let $\Pi(H)$ be an embedding of H on the plane.
Demoucron(G, H, $\Pi(H)$)

For ease of understanding and consistency, Algorithm 3.5 recomputes the faces and bridges from scratch at each recursive call. It is easy, however, to update both of these each time $\Pi(G')$ is modified - only the chosen bridge B and the face in which the path P is embedded need to be updated. If the latter is incorporated, we need only use the $O(n^2)$ face walking algorithm once to find the faces of the initial embedding of H. The algorithm would then have $O(m)$ or, equivalently, $O(n^2)$ running time. Obviously it is not necessary for Demoucron's algorithm to be recursive; Algorithm 3.5 gives non-recursive, $O(n^2)$ pseudocode for Demoucron's algorithm [15].

Algorithm 3.4 Democron(graph G, graph G', embedding $\Pi(G')$)

Use Algorithm 2.2 to find the faces of $\Pi(G')$.
Find the bridges of G with respect to G'.
if there are no bridges remaining **then**
  halt $\Pi(G')$ is an embedding of G.
else if there is a bridge with no admissible faces **then**
  halt G is not planar.
end if
Choose a bridge B of G with a minimum number of admissible faces.
Choose a bisecting path P of B.
Embed P in $\Pi(G')$.
Demoucron(G, G', $\Pi(G')$)

Algorithm 3.5 Demoucron_NR(graph G)

if G does not contain any cycles **then**
  return G is trivially planar.
end if
Choose a subgraph G' of G that is a cycle.
Let $\Pi(G')$ be an embedding of G' on the plane.
Use Algorithm 2.2 to find the faces of $\Pi(G')$.
Find the bridges of G with respect to G'.
while there are bridges remaining **do**
  if there is a bridge with no admissible faces **then**
    return G is not planar.
  end if
  Choose a bridge B of G with a minimum number of admissible faces.
  Choose a bisecting path P of B.
  Embed P in $\Pi(G')$, updating the bridges and faces.
end while
return $\Pi(G')$ is an embedding of G.

Chapter 4

The New Torus Embedding Algorithm

As does Demoucron's planar embedding algorithm, our new torus embedding algorithm follows the simple outline of the generic embedding framework. In this chapter we discuss the details and ways to further modify the generic framework to improve efficiency. Then we provide pseudocode for the algorithm and explain why it is correct.

4.1 Choice of Subgraph

First, if a graph is planar then it is also toroidal and a planar embedding of such a graph is also an embedding on the torus. We can use a planar embedding algorithm such as Demoucron's algorithm described in Section 3.2 to find out if a graph is planar and, if it is, find a planar embedding.

Recall from Theorem 2.3.1 that every nonplanar graph must contain a subgraph homeomorphic to either K_5 or $K_{3,3}$. Once we have established that our graph G is not planar, then, the first step in our torus embedding algorithm is to search for some subgraph H of G homeomorphic to K_5 or $K_{3,3}$. We can use a planar embedding algorithm to find such a subgraph in $O(n^3)$ time (if we use a linear time planar embedding algorithm) as shown by the pseudocode in Algorithm 4.1. More efficient algorithms are given by Klotz ($O(n^2)$) [28] and Williamson ($O(n)$) [41].

4.2 The Embeddings of K_5 and $K_{3,3}$ on the Torus

Up to isomorphism, there are six different unlabelled embeddings of K_5 and two different unlabelled embeddings of $K_{3,3}$ on the torus. These are shown in Figures 4.1

Algorithm 4.1 Find_K5_or_K33(graph G)

```
Let H = G
for all edges e in H do
  if H − e is not planar then
    H = H − e
  end if
end for
```

and 4.2, respectively.

Applying labels to these unlabelled embeddings of K_5 and $K_{3,3}$ in all possible nonequivalent ways yields labelled embeddings in which the faces differ. Our algorithm, therefore, must consider all *labelled* embeddings of a subgraph homeomorphic to K_5 or $K_{3,3}$ that it finds in a graph G to ensure it correctly ascertains whether or not G is toroidal.

The labelled embeddings were found by generating all nonequivalent orientable combinatorial embeddings for K_5 and $K_{3,3}$, and, for each of these embeddings:

- counting the faces using Algorithm 2.2, and
- using Euler's formula (Theorem 2.2.1) to check if the embedding has genus equal to two and is therefore a torus embedding [5].

On the torus, K_5 has 231 different labelled embeddings of and $K_{3,3}$ has 20 different labelled embeddings. Note that all of these embeddings divide the torus into faces that are homeomorphic to a planar disk.

4.3 Problems that Lead to Exponential Time

At first glance, it would appear that once we have divided the torus into faces by embedding a subgraph of G homeomorphic to K_5 or $K_{3,3}$ we should be able to proceed as with Demoucron's algorithm and find out whether the graph is embeddable on the torus in $O(n^2)$ time per embedding. There are two reasons why this is not the case.

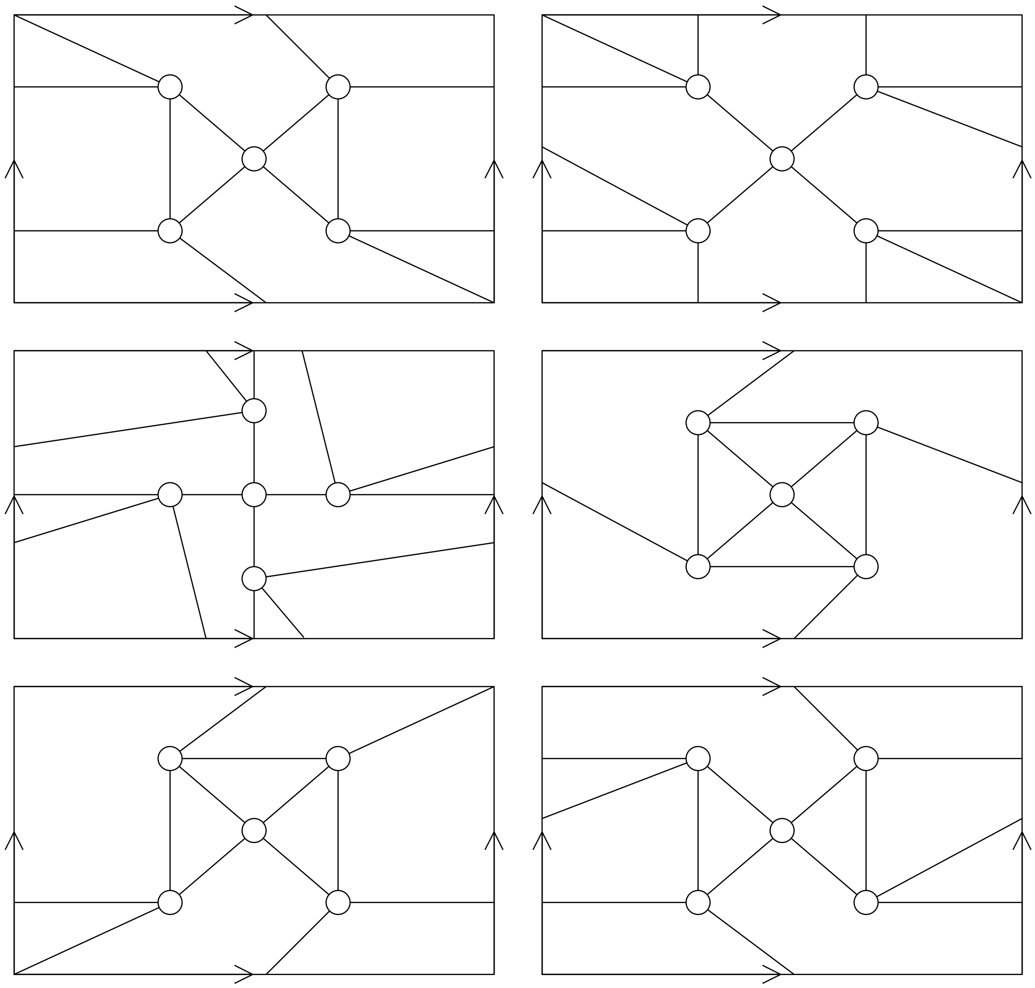


Figure 4.1: Unlabelled embeddings of K_5 on the torus.

First, unlike in Demoucron's algorithm, we cannot choose just one admissible face in which to embed a path from a bridge. Given a planar embedding of a subgraph G' of a graph G, if two bridges B_1 and B_2 of G with respect to G' both have admissible faces f_1 and f_2 and hinder each other with respect to f_1 then they must also hinder each other with respect to f_2. Thus no matter which face we choose for B_1, we decrease the number of possibilities for bridge B_2. On the torus, however, this argument does not hold. Figure 4.3 illustrates this: the dotted lines represent two bridges B_1 and

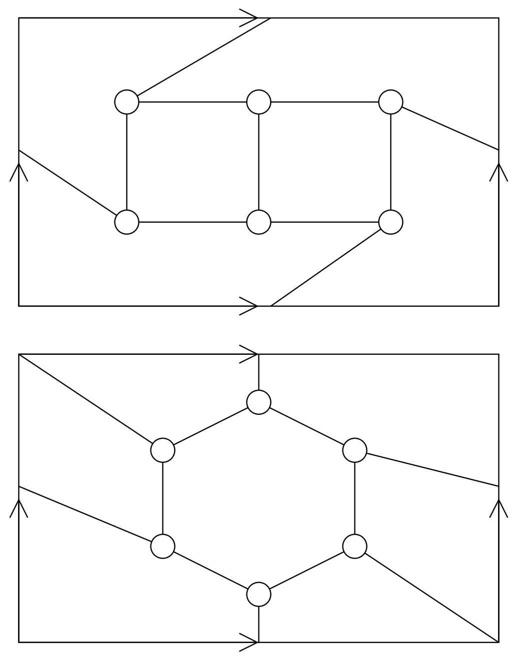


Figure 4.2: Unlabelled embeddings $K_{3,3}$ on the torus.

B_2 that hinder each other with respect to f_1 but not with respect to f_2. So, to avoid missing possible torus embeddings, after choosing a bridge B, we will embed a bisecting path from B in each of the admissible faces for B in turn.

The second reason we cannot proceed in $O(n^2)$ time per embedding with this approach is the presence of ugly faces in some of the embeddings of K_5 and $K_{3,3}$. Four of the unlabelled K_5 embeddings and one of the unlabelled $K_{3,3}$ embeddings yield labelled embeddings with ugly faces as illustrated in Figures 4.4 and 4.5, respectively.

Notice that some of the ugly faces of the K_5 and $K_{3,3}$ embeddings contain repeated edges as well as repeated vertices. In Figure 4.5 for example, edges $(1,6)$ and $(3,4)$ are repeated. Subdividing these repeated edges creates more repeated vertices on the ugly face. Thus the embeddings of graphs homeomorphic to K_5 or $K_{3,3}$ might have ugly faces with more repeated vertices than the corresponding K_5 or $K_{3,3}$ embedding.

Because of the presence of repeated vertices on the ugly faces, there might be

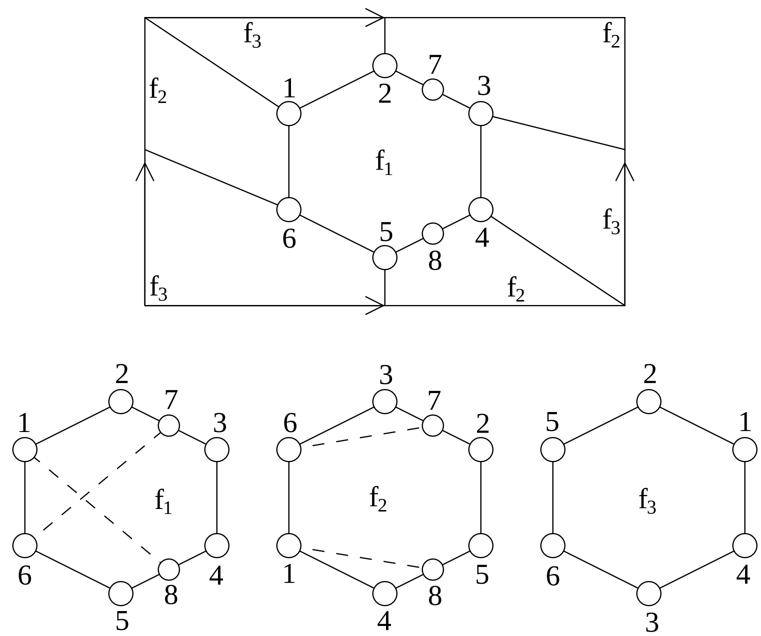


Figure 4.3: An embedding of a $K_{3,3}$ homeomorph and its three faces, along with two bridges that hinder each other with respect to f_1 but not with respect to f_2.

more than one way to connect the internal vertices of a bridge B to its attachment vertices if we are embedding B in an ugly face. Figure 2.7 showed a bridge that is only embeddable in an ugly face f if both copies of two repeated vertices of f are used. Also, it is possible for two bridges to hinder each other with respect to ugly face f if one combination of attachment vertices is chosen, and not hinder each other with respect to f if another combination of attachment vertices is chosen. The left-hand side of Figure 4.6 shows an embedding of a graph containing a subgraph homeomorphic to $K_{3,3}$, and two bridges that are both embedded in the ugly face, f, of the $K_{3,3}$. The right-hand side of Figure 4.6 shows that the two bridges hinder one another if the wrong copy of vertex 3 is chosen to attach to vertex v. If the solid edge between vertices v and 3 were replaced by the dotted one, the two bridges would no longer hinder one another.

Although there is no limit to the number of repeated vertices on the ugly face of an embedding of a graph homeomorphic to K_5 or $K_{3,3}$, it is clear that each repeated vertex can occur at most twice on such a face. Further, the maximum number of times a vertex is repeated on some face cannot increase when a bisecting path is embedded across a face. Therefore, there can be at most four possible ways to embed a bisecting path from a bridge in a face. In fact, if a and b are the endpoints of a bisecting path P from a bridge B, and x_a and x_b equal the number of times a and b, respectively, occur on a face f that is admissible for B, then P can be embedded $x_a \cdot x_b$ ways in f.

4.4 Choosing a Bridge

At each stage of recursion, the choice of the bridge from which we embed a bisecting path can significantly reduce the size of the recursion tree. It is clear, for example, that if we could determine that there is only one way to embed a bridge that we should choose this bridge and embed it. In doing so, we might eliminate some (possibly large) branches of the recursion tree of our algorithm. To assist our algorithm in making a sensible choice of bridge, then, we define a penalty $P(B)$ for each bridge B as follows. For each admissible face f for B:

- let x_i be the number of times attachment vertex i of B appears on f, and
- choose two different attachment vertices u_f and v_f of B such that $x_{u_f} \cdot x_{v_f}$ is minimized.

Now,

$$P(B) = \sum_{f \textit{ is admissible for } B} x_{u_f} \cdot x_{v_f}.$$

If there is a bridge B with $P(B) = 0$, our algorithm must backtrack as there is a bridge that has no admissible faces. Otherwise, it chooses a bridge B with minimum penalty. Further, for each admissible face f for B we can choose a bisecting path between the vertices u_f and v_f that were chosen when computing the penalty for B. Such a path can be embedded $x_{u_f} \cdot x_{v_f}$ ways in f and this must be minimum over all paths in B because of the way u_f and v_f were chosen. In this way we select a bridge that minimizes the number recursive calls at each stage of recursion; by doing so we significantly improve the running time of our algorithm in most cases.

4.5 Pseudocode for the New Algorithm

Algorithms 4.2 and 4.3 give pseudocode for the new algorithm.

Algorithm 4.2 StartTorusEmbed(graph G)

if G is planar **then**
    halt a planar embedding of G is also a torus embedding of G.
else
    Choose a subgraph H of G that is homeomorphic to either K_5 or $K_{3,3}$.
    for all non-isomorphic, nonequivalent labelled embeddings $\Pi(H)$, of H **do**
        TorusEmbed(G, H, $\Pi(H)$)
    end for
end if

4.6 Correctness

It is not difficult to see that our new algorithm correctly determines if a graph is toroidal and as such no complicated proof is necessary. By Theorem 2.3.1 it is clear that any graph G that is not planar contains a subgraph H homeomorphic to either K_5 or $K_{3,3}$. The algorithm considers all possible embeddings of H on the torus and all possible ways of embedding the bridges in each embedding. Our algorithm finds an embedding of G if and only if G is toroidal.

Algorithm 4.3 TorusEmbed(graph G, graph G', embedding $\Pi(G)$)

Use Algorithm 2.2 to find the faces of $\Pi(G')$.
Find the bridges of G with respect to G' and the penalty $P(B)$ for each bridge.
if there are no bridges remaining **then**
  halt $\Pi(G')$ is an embedding of G.
end if
if there is a bridge B with $P(B) = 0$ **then**
  return $\Pi(G')$ cannot lead to an embedding of G (backtrack).
end if
Choose a bridge B with minimum $P(B)$.
for all admissible faces, f, for B **do**
  Choose a bisecting path P from B with endpoints u_f and v_f (see Section 4.4).
  Let u_{f2} and v_{f2} be the second copies of vertices u_f and v_f on face f, respectively, if they exist.
  Embed P in $\Pi(G')$ using endpoints u_f and v_f.
  TorusEmbed(G,G',$\Pi(G)$)
  Remove P from $\Pi(G')$.
  if vertex u_f is repeated on face f **then**
    Embed P in $\Pi(G')$ using endpoints u_{f2} and v_f.
    TorusEmbed(G,G',$\Pi(G)$)
    Remove P from $\Pi(G')$.
  end if
  if vertex v_f is repeated on face f **then**
    Embed P in $\Pi(G')$ using endpoints u_f and v_{f2}.
    TorusEmbed(G,G',$\Pi(G)$)
    Remove P from $\Pi(G')$.
  end if
  if vertices u_f and v_f are both repeated on face f **then**
    Embed P in $\Pi(G')$ using endpoints u_{f2} and v_{f2}.
    TorusEmbed(G,G',$\Pi(G)$)
    Remove P from $\Pi(G')$.
  end if
end for

4.7 Bad Input Graphs: An Afterthought

Initial timing studies of our algorithm led us to discover graphs which caused our algorithm to be very slow. Analysis of the structure of such graphs revealed that the slowness was a result of having multiple ways to embed some bridges when the graph had one or more 2-vertex cuts. It is possible, however, because of Theorem 4.7.1 below, to preprocess these graphs to avoid the slow running time when they are given as input to our algorithm.

Theorem 4.7.1. *Let B be a bridge of a graph G with respect to the subgraph consisting of the two vertices a and b (and no edges) of a 2-vertex cut $\{a, b\}$ in G. If $B + (a, b)$ is planar, then G embeds on S if and only if*

$$G - \{v|v \text{ } is \text{ } an \text{ } internal \text{ } vertex \text{ } of \text{ } B\} + (a, b)$$

embeds on S.

Proof. Given an embedding of

$$G - \{v|v \text{ } is \text{ } an \text{ } internal \text{ } vertex \text{ } of \text{ } B\} + (a, b),$$

we can replace (a, b) with a planar embedding of B or, if (a, b) is an edge in G, replace (a, b) with a planar embedding of $B + (a, b)$. $\square$

We created a preprocessor to reduce to a single edge (a, b) any bridge B with respect to some 2-vertex cut $\{a, b\}$ such that $B + (a, b)$ is planar. This significantly reduced the running time on input graphs containing such bridges.

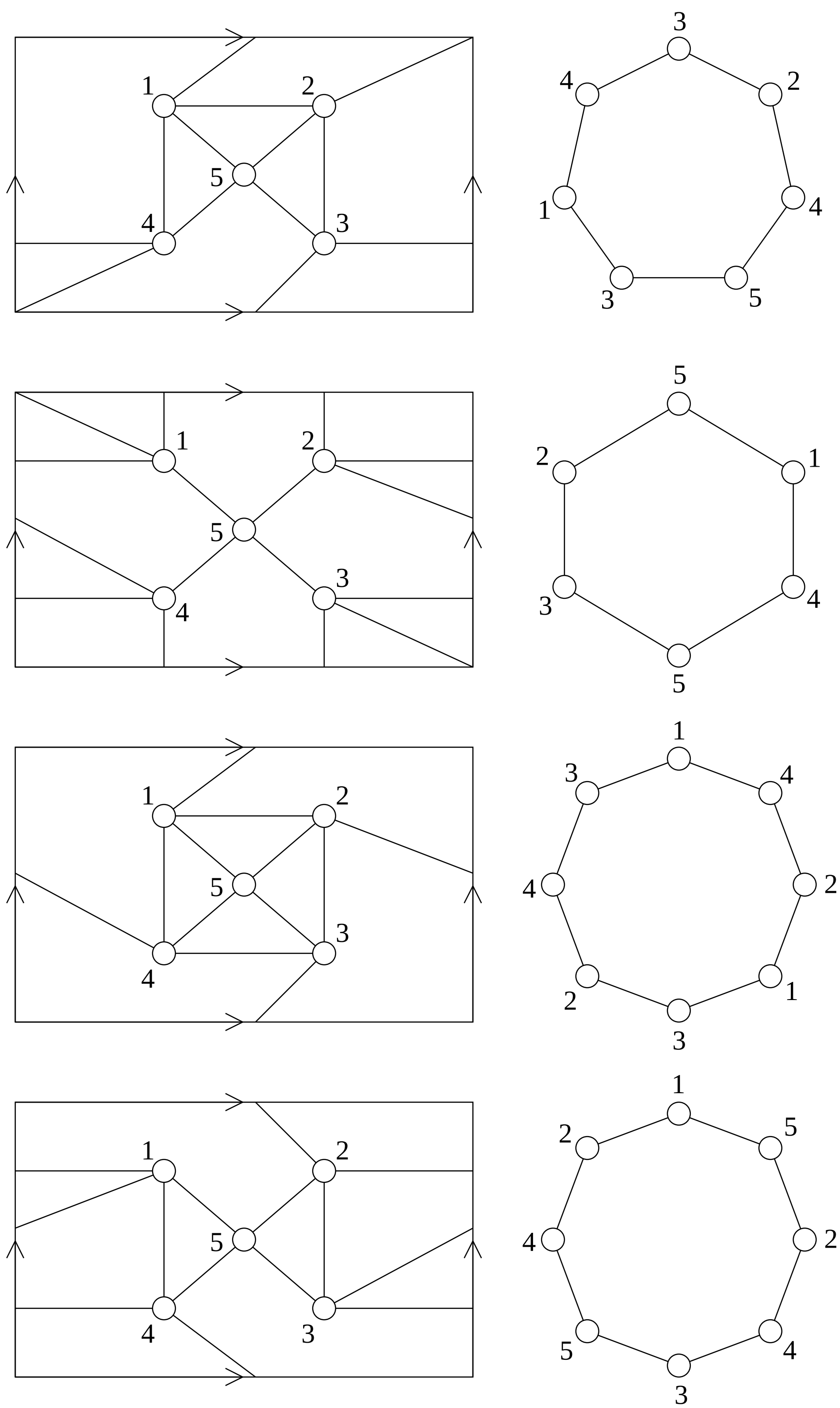


Figure 4.4: Ugly faces of four of the K_5 embeddings.

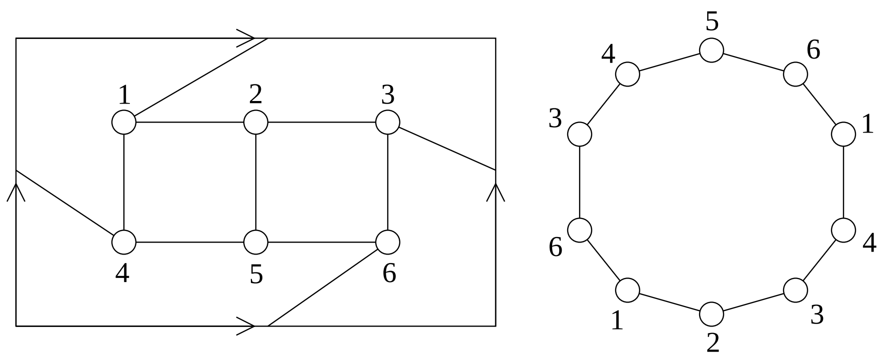


Figure 4.5: Ugly face of one of the $K_{3,3}$ embeddings.

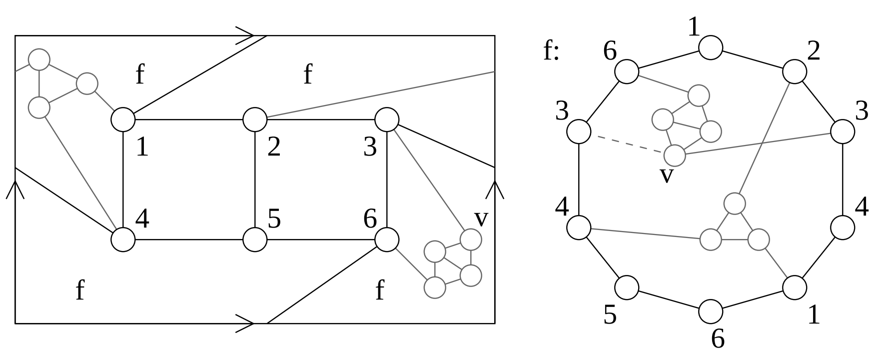


Figure 4.6: Two possibilities for a pair of bridges in the ugly face of $K_{3,3}$.

Chapter 5

Computational Results

We implemented our new torus embedding algorithm in C and have performed several timing comparisons with the previous torus embedding algorithm of Myrvold and Neufeld [34, 33], also implemented in C. Both pieces of code were run on the same computer with an Intel Pentium 4, 3.6GHz processor. The following sections present these comparisons. The timings are given in milliseconds and insignificant times - below 1 millisecond - are indicated by a 0 in the tables.

5.1 Known Obstructions

Algorithm 2.3 showed how a torus embedding algorithm can be used to check if a graph is a topological or minor-order torus obstruction. If the input graph G *is* a torus obstruction, it is in some sense "almost" toroidal, since $G - e$ (and $G \cdot e$ in the case of a minor-order obstruction) is toroidal for any edge e in G. Thus it seems intuitively possible that a torus embedding algorithm would have to work harder to find that these graphs are not toroidal; of course this depends on the structure of the graph and how efficient the algorithm is for graphs with such structure. Further, since we know that $G - e$ (and $G \cdot e$ in the case of a minor-order obstruction) is toroidal for every edge e in G, the torus embedding algorithm is called at least $m + 1$ times for a topological obstruction with m edges and $2m + 1$ times for a minor-order obstruction with m edges.

We timed the two algorithms as they confirmed that the 239,451 known torus obstructions computed in [34, 33, 12] are in fact obstructions and checked whether or not they are minor-order obstructions. These graphs have $8 \leq n \leq 24$ and $\lceil 3n/2 \rceil \leq$

		Old Algorithm				New Algorithm			
n	#	Min	Max	Avg	Total	Min	Max	Avg	Total
8	3	250	2,370	1,240	3,720	80	170	113	340
9	48	10	4,070	1,012	48,610	0	320	93	4,510
10	660	0	7,420	933	616,330	0	540	81	53,620
11	4,923	0	16,450	648	3,193,680	0	3,270	61	300,700
12	18,458	0	4,330	452	8,355,160	0	2,390	47	883,180
13	38,466	0	3,430	320	12,320,510	0	3,160	38	1,499,550
14	61,343	30	2,520	255	15,647,490	0	920	32	2,002,480
15	57,434	50	1,940	214	12,322,840	0	540	30	1,723,770
16	35,672	40	1,920	199	7,128,940	10	420	28	1,026,830
17	15,564	60	1,160	204	3,189,690	10	300	31	497,750
18	5,168	60	1,050	232	1,200,740	20	260	35	183,910
19	1,390	100	1,210	283	393,550	20	210	39	55,310
20	224	110	690	315	70,750	30	60	39	8,920
21	68	240	530	343	23,380	30	60	43	2,970
22	24	260	450	353	8,480	40	60	47	1,140
23	4	310	410	370	1,480	40	50	47	190
24	2	330	350	340	680	50	60	55	110
8-24	239,451	0	16,450	269	64,526,030	0	3,270	34	8,245,280

(Times in Milliseconds)

Table 5.1: Results of using the 239,451 known torus obstructions as input.

$m \leq 3n + 1$. Table 5.1 shows the results and Figure 5.1 shows a plot of the average running times of the two algorithms. On average, the new algorithm is approximately eight times faster than the old algorithm to decide that a graph is a topological obstruction and whether or not it is also a minor-order torus obstruction.

5.2 Random Large Graphs

To test the efficiency of our new algorithm on larger graphs, we implemented a random graph generator to create toroidal and nontoroidal graphs with a desired number of vertices. The generator begins by randomly choosing an embedding of either K_5 or $K_{3,3}$ on the torus and then repeatedly and randomly chooses one of the following graph updates until the desired order is reached.

Update 1: Choose two different edges (a, b) and (c, d) on the same face f of the embedding. Subdivide each edge to create the edges (a, x), (x, b), (c, y), and

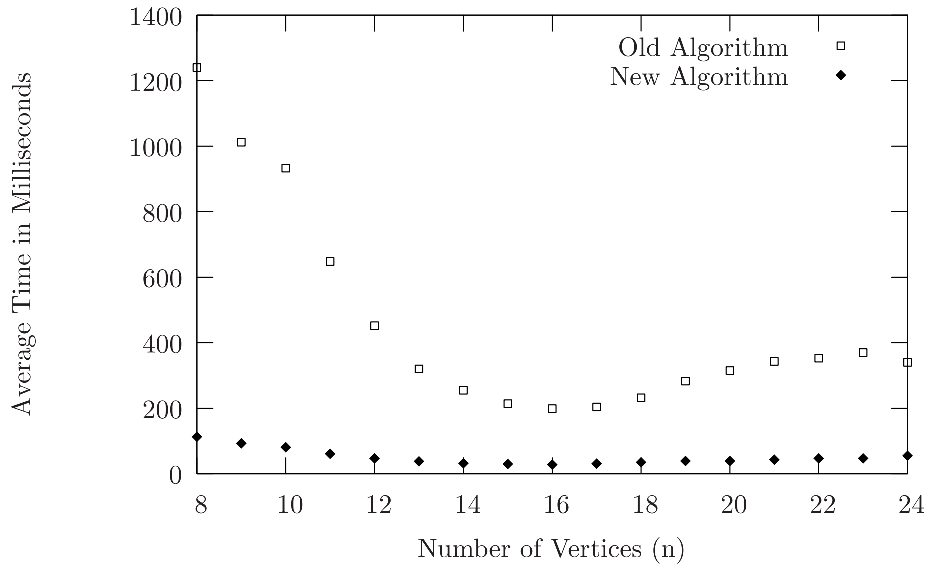


Figure 5.1: Plot of the average running times for all known obstructions.

(y, d) and remove the edges (a, b) and (c, d). Then add the edge (x, y) thereby splitting f into two faces. This increases the number of vertices by two.

Update 2: Choose one edge (a, b) and one vertex c on the same face f such that $b \neq c$ and $a \neq c$. Subdivide (a, b) to create the edges (a, x) and (x, b), and remove the edge (a, b). Then add the edge (x, c) thereby splitting f into two faces. This increases the number of vertices by one.

Update 3: Choose two different vertices a and b on the same face f such that (a, b) is not already an edge of the graph. Then add the edge (a, b) thereby splitting f into two faces. This does not increase the number of vertices.

Once we have the desired number of vertices in the graph, we either output the graph as a random toroidal graph, or randomly add edges to the graph until it is not toroidal (using our new algorithm to test this) and then output the graph as a random nontoroidal graph.

		Old Algorithm				New Algorithm			
n	#	Min	Max	Avg	Total	Min	Max	Avg	Total
10	100	0	10	0	40	0	10	0	10
20	100	0	40	4	490	0	10	1	100
30	100	0	150	22	2,210	0	20	3	390
40	100	0	1,340	90	9,000	0	70	10	1,010
50	100	0	2410	248	24,860	0	230	19	1,910
60	100	10	2,360	449	44,920	0	410	31	3,140
70	100	10	4,840	703	70,340	10	2,010	69	6,970
80	100	30	26,480	1,656	165,600	20	3,480	92	9,250
90	100	50	64,720	3,679	367,950	20	77,810	923	92,380
100	100	70	57,050	3,529	352,930	30	1,390	145	14,540
110	100	80	276,180	8,579	857,930	40	11,650	351	35,150
120	100	130	89,110	7,814	781,440	40	4,060	361	36,100
130	100	110	119,460	10,722	1,072,200	70	17,860	891	89,140
140	100	220	110,060	15,996	1,599,680	80	36,480	1,205	120,590
150	100	250	392,970	21,015	2,101,530	100	4,790	544	54,420
160	100	190	846,200	34,873	3,487,380	110	30,230	1,460	146,010
170	100	290	2,298,510	57,860	5,786,050	120	1,084,040	11,946	1,194,660
180	100	450	530,000	61,183	6,118,390	170	3,193,980	37,474	3,747,470
190	100	600	1,254,340	81,236	8,123,660	210	5,532,240	69,239	6,923,960
200	100	510	1,253,850	82,029	8,202,930	190	378,110	12,586	1,258,690
10-200	2,000	0	2,298,510	19,584	39,169,530	0	5,532,240	6,867	13,735,890

(Times in Milliseconds)

Table 5.2: Results using small randomly generated toroidal graphs as input.

In these ways, 100 toroidal graphs for $n = 10, 20, \ldots, 200$ and 100 nontoroidal graphs for $n = 10, 20, \ldots, 140$ were generated giving 2000 random toroidal graphs and 1400 random nontoroidal graphs. The timing data for the two algorithms using these graphs as input is given in Tables 5.2 and 5.3 and plots of the average running times for both algorithms are shown in Figures 5.2 and 5.3. The new algorithm ran approximately three times faster with the toroidal graphs as input and 465 times faster with the nontoroidal graphs as input.

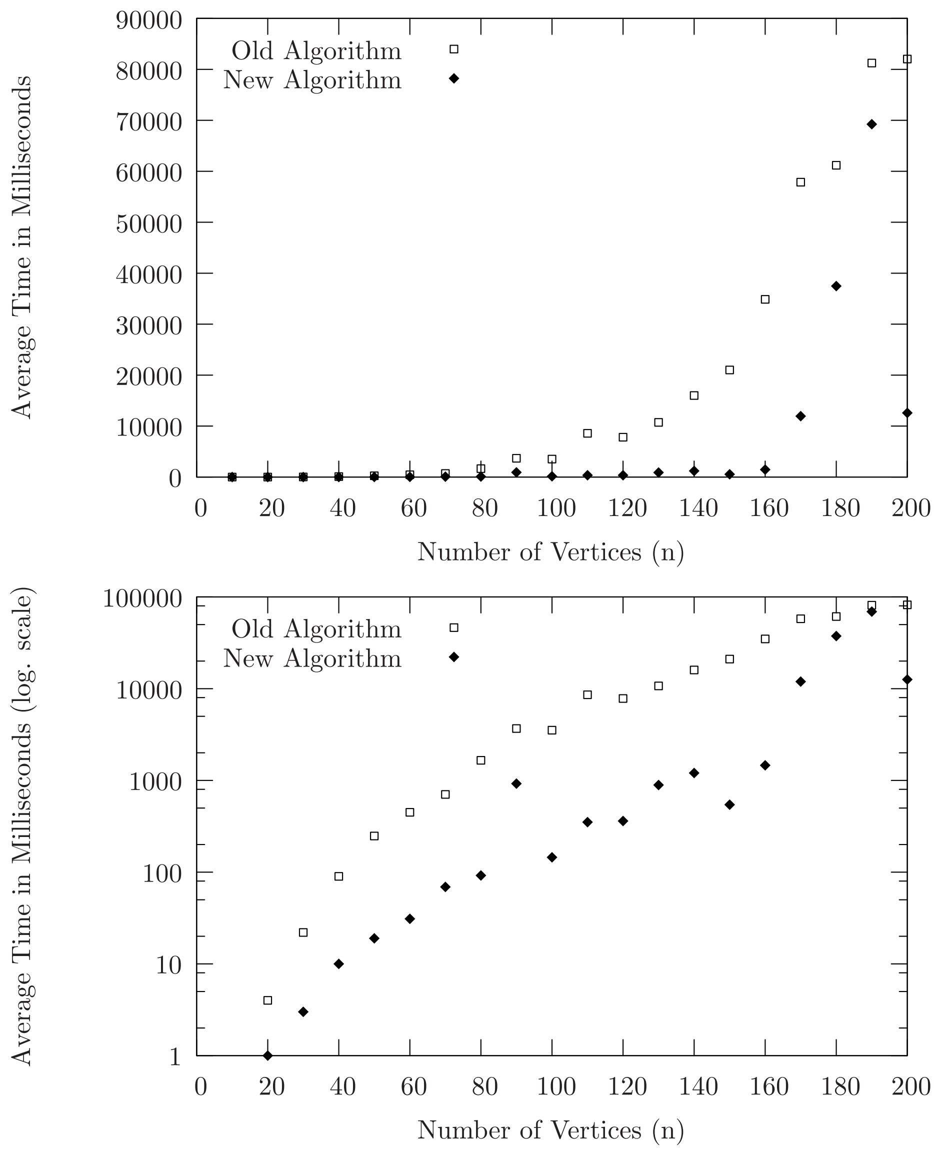


Figure 5.2: Plots of the average running times for small randomly generated toroidal graphs.

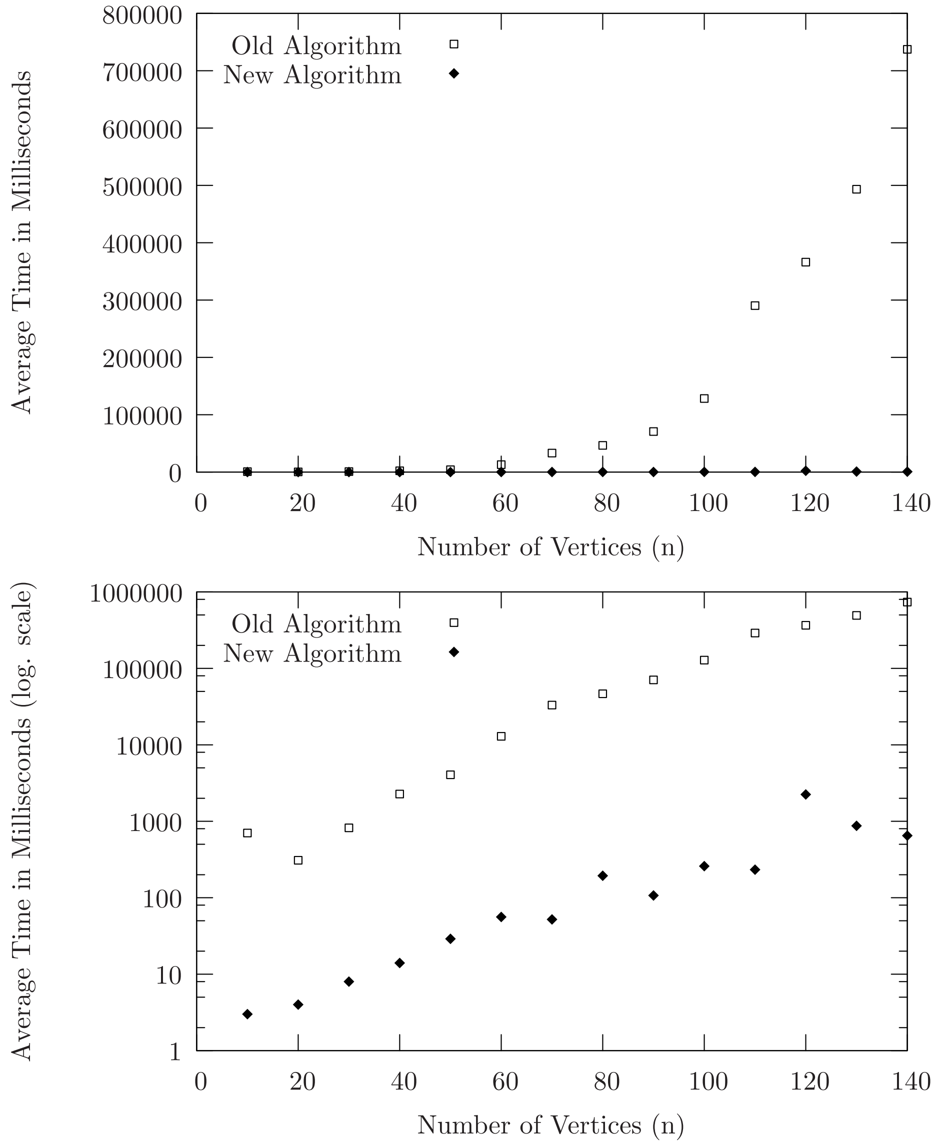


Figure 5.3: Plots of the average running times for small randomly generated non-toroidal graphs.

		Old Algorithm				New Algorithm			
n	#	Min	Max	Avg	Total	Min	Max	Avg	Total
10	100	70	2,870	702	70,290	0	20	3	370
20	100	50	1,290	309	30,980	0	30	4	410
30	100	130	5,790	821	82,130	0	90	8	850
40	100	370	13,840	2,272	227,230	0	100	14	1,490
50	100	650	41,680	4,057	405,780	0	290	29	2,900
60	100	850	99,440	12,929	1,292,950	10	820	56	5,680
70	100	3,050	608,680	33,144	3,314,440	10	340	52	5,200
80	100	4,500	424,300	46,550	4,655,060	20	9,470	194	19,490
90	100	6,710	434,460	70,732	7073270	20	750	107	10,700
100	100	10,410	1,194,340	128,281	12,828,120	30	3,380	259	25,980
110	100	15,980	3,262,680	290,312	29,031,280	40	1,630	233	23,320
120	100	16,050	6,022,320	366,148	36,614,890	40	108,510	2244	224,440
130	100	26,360	9,279,460	493,267	49,326,740	70	24,640	872	87,210
140	100	44,470	10,815,250	737,345	73,734,570	50	9,290	649	64,920
10-140	1,400	50	10,815,250	156,205	218,687,730	0	108,510	337	472,960

(Times in Milliseconds)

Table 5.3: Results using small randomly generated nontoroidal graphs as input.

Chapter 6

Conclusions and Future Research

In this thesis, we have presented a new exponential algorithm for embedding graphs on the torus that was inspired by Demoucron's $O(n^2)$ algorithm for embedding graphs on the plane. We explained the differences in our approach for the torus that lead to its exponential running time but gave results to show that, although it is exponential, it is faster in practice than a previous exponential algorithm that was used to find many torus obstructions. In this final section, we discuss two ways in which the running time of our algorithm might be improved and we revisit the problem of searching for torus obstructions that provided motivation for developing our new algorithm.

6.1 Enhancing Our Algorithm

Since there are 231 non-isomorphic labelled embeddings of K_5 and only 20 non-isomorphic labelled embeddings of $K_{3,3}$ on the torus, it would be preferable if the subgraph which our algorithm initially embeds on the torus were homeomorphic to $K_{3,3}$. Given an input graph G and a subgraph G' of G homeomorphic to K_5 it is possible to either:

- find a subgraph homeomorphic to $K_{3,3}$ in G, if one exists, or
- perform a small constant number of planarity tests to determine if G is toroidal if G has no subgraph homeomorphic to $K_{3,3}$ [19].

The "transformation" of K_5 to $K_{3,3}$ can be done in linear time using the method of Asano [6]. Incorporating this transformation into our algorithm would not improve upon its exponential running time. However, we expect that it would decrease the running time in practice in most cases where it initially finds a subgraph homeomorphic to K_5 in G. The cases where this might not be true include:

- when an embedding could be found using one of the first twenty embeddings of K_5 that would have been considered, and
- when the invalid embeddings of the K_5 would have been quickly rejected.

In the cases where the graph does not contain a subgraph homeomorphic to $K_{3,3}$ and therefore the transformation is not successful the test for toroidality can be performed in linear time [19] using a linear time planar embedding algorithm. Modifying our algorithm to take this approach when the input graph does not contain a subgraph homeomorphic to $K_{3,3}$, then, would make our algorithm run in linear time for these graphs. Given the importance of the search for a complete set of torus obstructions in motivating the development of our algorithm, it is necessary to point out that in an exhaustive search for more torus obstructions, we do not need to consider graphs that do not contain a subgraph homeomorphic to $K_{3,3}$. This is because all torus obstructions with this property have already been found [13].

Our algorithm would also likely be more efficient if it separated the two copies of repeated vertices on ugly faces as often and as early as possible. More formally, suppose that we have embedded a subgraph G' of graph G and that the embedding has some ugly face f with k repeated vertices. Consider a bridge B of G with respect to G' for which f is admissible and let k' and k'' be the number of repeated vertices on the two faces that result from embedding some bisecting path P in f. Currently, for face f, our algorithm chooses as endpoints of P attachment vertices u_f and v_f that minimize the value of $x_{u_f} \cdot x_{v_f}$ (as defined in section 4.4). The efficiency of our algorithm could be improved in many cases if it also chose, from among the pairs of vertices u and v that minimize $x_u \cdot x_v$, a pair of vertices u_f and v_f that minimize the value of $k'+k''$. To illustrate this, in Figure 6.1, the graph on the left-hand side shows the ugly face of an embedding and one bridge. The two right-hand graphs show the resulting faces if a bisecting path is embedded between vertices 2 and 5 (in which case neither of the faces has a repeated vertex) and between vertices 2 and 7 (in which case one face has two repeated vertices and the other has none). It would clearly be better to embed the path between vertices 2 and 5 in this example. Efficiency could be even further improved if, from among the bridges that have minimum penalty, our algorithm selected a bridge that would minimize $k' + k''$.

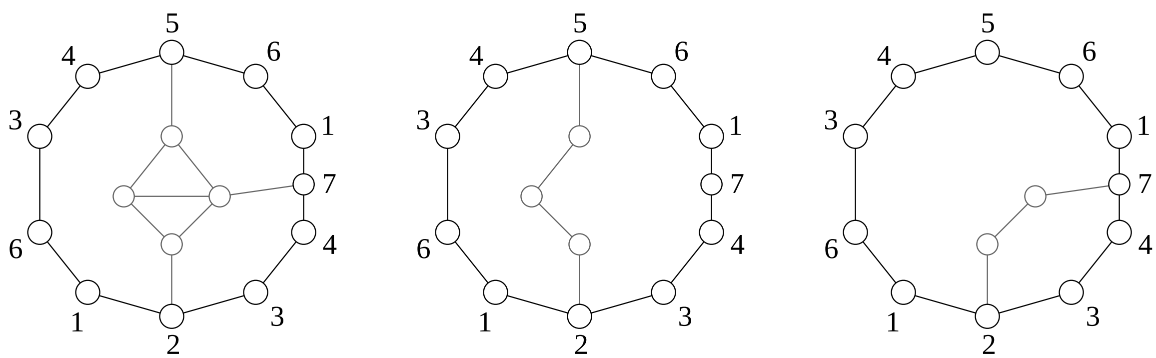


Figure 6.1: Using repeated vertices to embed a bridge in a face.

6.2 Searching for Torus Obstructions

As mentioned in the introduction, finding the complete set of obstructions to the torus would be a major breakthrough in topological graph theory. Currently, the major bottleneck in achieving the overall goal is a lack of stopping criteria for a computer search; we do not yet have an upper bound on the number of vertices a torus obstruction can have.

Even when we know torus obstructions of order n exist, there are values of n for which there are simply too many graphs to be able to do an exhaustive search for obstructions without much faster torus embedding code or a way of limiting the structure of the graphs that are obstruction candidates. As discussed in Section

2.3 of this thesis, attempts to design and implement a fast enough torus embedding algorithm to do such exhaustive searches have not been fruitful. So it seems we must turn our attention to reducing the number of candidate graphs so that the algorithm presented in this thesis can complete the search.

Having a large database of known torus obstructions provides the opportunity to analyze the structure of these graphs and make and prove conjectures about the structure of the complete set. Since all torus obstructions are nonplanar and all projective planar torus obstructions have been found [25], one approach is to examine the structure of known torus obstructions with respect to a subgraph which is an obstruction for the plane or projective plane. This technique has already led to successful characterization of the torus obstructions which do not contain a subgraph homeomorphic to $K_{3,3}$ (as mentioned in section 6.1) [13]. Another approach is to choose some subclass of the known obstructions and analyze their structure with the aim of characterizing all of the obstructions that belong to that class. The obstructions that have a 2-vertex cut form a promising choice of subclass for this approach.

We hope that these theoretical analyses and characterizations eventually lead us to determine an upper bound, N, on the order of a torus obstruction. If the set of torus obstructions still has not been proved complete at this point, we believe that sufficient structural characterizations can limit the number of obstruction candidates enough to perform exhaustive searches using our torus embedder on graphs of orders twelve through N. Alternatively, structural characteristics might be used to define ways of generating all torus obstructions from scratch. In any case, despite the daunting number of torus obstructions that have already been found, we believe that there are methods to finish the search for the complete set.

Bibliography

[1] D. Archdeacon. A Kuratowski theorem for the projective plane. *Journal of Graph Theory*, 5:243–246, 1981.

[2] D. Archdeacon and J. P. Huneke. A Kuratowski theorem for nonorientable surfaces. *Journal of Combinatorial Theory. Series B*, 46:173–231, 1989.

[3] Dan Archdeacon. Topological graph theory: a survey. *Congressus Numerantium. A Conference Journal on Numerical Themes*, 115:5–54, 1996.

[4] Dan Archdeacon and C. Paul Bonnington. Obstructions for embedding cubic graphs on the spindle surface. *Journal of Combinatorial Theory. Series B*, 91(2):229–252, 2004.

[5] Angus Argyle. Toroidal embeddings of $K_{3,3}$ and K_5. Honours Project Technical Report, supervised by Wendy Myrvold.

[6] Takao Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38(2-3):249–267, 1985.

[7] Joseph Battle, Frank Harary, Yukihiro Kodama, and J.W.T. Youngs. Additivity of the genus of a graph. *American Mathematical Society Bulletins*, 68:565–568, 1962.

[8] R. Bodendiek and K. Wagner. Solution to König's graph embedding problem. *Mathematische Nachrichten*, 140:251–272, 1989.

[9] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. American Elsevier Publishing Co., Inc., New York, 1976.

[10] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[11] 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*, pages 140–146, 1999.

[12] J. Chambers. Hunting for torus obstructions. Master's thesis, Department of Computer Science, University of Victoria, Victoria, B.C., 2002.

[13] J. Chambers, A. Gagarin, and W. Myrvold. The obstructions for toroidal graphs with no $K_{3,3}$'s. 2004 preprint. 13 pages.

[14] H. de Fraysseix and P. Rosentiehl. A depth-first search characterization of planarity. *Ann. Discrete Math.*, 13:75–80, 1982.

[15] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires. *Revue Francaise Recherche Operationnelle*, 8:33–47, 1964.

[16] I. S. Filotti. An algorithm for imbedding cubic graphs in the torus. *Journal of Computer and System Sciences*, 2:255–276, 1980.

[17] Patrick W. Fowler. Complexity, spanning trees and relative energies in fullerene isomers. *MATCH. Communications in Mathematical and in Computer Chemistry*, (48):87–96, 2003. Dedicated to Professor Horst Sachs on the occasion of his 75th birthday.

[18] Patrick W. Fowler, Kevin M. Rogers, Siemion Fajtlowicz, Pierre Hansen, and Gilles Caporossi. Facts and conjectures about fullerene graphs: leapfrog, cylinder and Ramanujan fullerenes. In *Algebraic combinatorics and applications (Gößweinstein, 1999)*, pages 134–146. Springer, Berlin, 2001.

[19] Andrei Gagarin and William Kocay. Embedding graphs containing K_5-subdivisions. *Ars Combinatoria*, 64:33–50, 2002.

[20] A. Galluccio and M. Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6:1–18, 1999.

[21] Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[22] H. Glover, J. Huneke, and C. Wang. 103 graphs that are irreducible for the projective plane. *Journal of Combinatorial Theory Series B*, 27:332–370, 1979.

[23] Michael Henle. *A Combinatorial Introduction to Topology*. Dover Publications, Inc., New York, 1994.

[24] J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[25] M. Juvan. *Algorithms and obstructions for embedding graphs in the torus*. PhD thesis, University of Ljubljana, 1995. (in Slovene).

[26] M. Juvan, J. Marincek, and B. Mohar. Embedding graphs in the torus in linear time. *Lecture Notes in Computer Science*, 920:360–363, 1995.

[27] M. Juvan and B. Mohar. A simplified algorithm for embedding graphs in the torus. Preprint, 10 pages. URL: http://www.fmf.uni-lj.si/ mohar/Papers/Torus.pdf, 2001.

[28] W. Klotz. A constructive proof of Kuratowski's theorem. *Ars Combinatoria*, 28:51–54, 1989.

[29] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[30] B. Mohar. Projective planarity in linear time. *Journal of Algorithms*, 15:482–502, 1993.

[31] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2001.

[32] Wendy Myrvold and William Kocay. Errors in graph embedding algorithms. In preparation. Presented at the Thirty-fifth Southeastern International Conference on Combinatorics, Graph Theory, and Computing, 2004.

[33] E. Neufeld. Practical toroidality testing. Master's thesis, Department of Computer Science, University of Victoria, Victoria, B.C., 1993.

[34] E. Neufeld and W. Myrvold. Practical toroidality testing. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 574–580, 1997.

[35] N. Robertson and P. Seymour. Graph minors. VIII. a Kuratowski theorem for general surfaces. *Journal of Combinatorial Theory. Series B*, 48:255–288, 1990.

[36] Jianping Roth and Wendy Myrvold. Simpler projective plane embedding. *Ars Combinatoria*, 75:135–155, 2005.

[37] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[38] K. Wagner. Über einer eigenschaft der ebener complexe. *Mathematische Annalen*, 114:570–590, 1937.

[39] Douglas B. West. *Introduction to graph theory*. Prentice Hall Inc., Upper Saddle River, NJ, 1996.

[40] S. G. Williamson. Embedding graphs in the plane- algorithmic aspects. *Annals of Discrete Mathematics*, 6:349–384, 1980.

[41] S. G. Williamson. Depth-first search and Kuratowski subgraphs. *Journal of the ACM*, 31(4):681–693, 1984.

Partial Copyright License