Simpler Projective Plane Embedding

Jianping Roth
Creo Inc., Vancouver

Wendy Myrvold *
University of Victoria
wendym@cs.UVic.ca

Abstract

A *projective plane* is equivalent to a disk with antipodal points identified. A graph is *projective planar* if it can be drawn on the projective plane with no crossing edges. A linear time algorithm for projective planar embedding has been described by Mohar. We provide a new approach that takes $O(n^2)$ time but is much easier to implement. We programmed a variant of this algorithm and used it to computationally verify the known list of all the projective plane obstructions.

One application for this work is graph visualization. Projective plane embeddings can be represented on the plane and can provide aesthetically pleasing pictures of some non-planar graphs. More important is that it is highly likely that many problems that are computationaly intractible (for example, NP-complete or #P-complete) have polynomial time algorithms when restricted to graphs of fixed orientable or non-orientable genus. Embedding the graph on the surface is likely to be the first step for these algorithms.

1 Background

A *graph* G consists of a set V of vertices and a set E of edges, each of which is associated with an unordered pair of vertices from V. Throughout this paper, n denotes the number of vertices of a graph, and m is the number of edges. A graph is *embeddable* on a surface M if it can be drawn on M without crossing edges. Archdeacon's survey [2] provides an excellent introduction to topological graph theory (the study of graph embeddings) and includes many interesting open questions.

A graph can be used to model many things. Some examples with applications in computer science include modelling program structure, networks, or how documents on the web are linked together using hyperlinks. A graph

*Supported by NSERC.

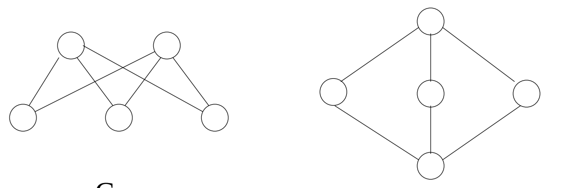


Figure 1: A graph and a planar embedding

visualization tool can help researchers to better understand the structure of such things. Usually, it is best to avoid having many crossing edges as this can complicate the picture of a graph. Algorithms for embedding graphs on surfaces are often used to help get a nice picture of a graph.

A graph is *planar* if it can be embedded on the plane. A *planar embedding* of a graph is a description of how it can be embedded in the plane. A graph and one planar embedding of it are pictured in Figure 1. A planar embedding is commonly represented by giving the clockwise order of the neighbours of each vertex (the *combinatorial embedding*). This paper is concerned with combinatorial embeddings. The aesthetic issues of where to place the vertices and edges are discussed in a wide body of literature including the recent book by Battista, Eades, Tamassia and Tollis [4]. It is well-known that a planar graph without loops or multiple edges has at most $3n - 6$ edges. Thus in discussing time complexities for algorithms, *linear time* should be interpreted as time in $O(n)$.

There are several linear time algorithms for finding a planar embedding of a graph. The first of these was developed in 1974 by Hopcroft and Tarjan [21]. Booth and Lueker derived another approach which uses a data structure called a PQ-tree [6]. Other algorithms include those of de Fraysseix and Rosentiehl [11] and Williamson [35, 36]. More recently, Boyer and Myrvold have proposed a new approach [7]. All of these algorithms are fairly complex. Two slower approaches to planarity testing that are easier to program are the $O(n^2)$ algorithms of Demoucron, Malgrange, and Pertuiset [12] and Klotz [23].

A *torus* is the surface created by attaching one handle to the sphere. A simple toroidal graph can have at most $3n$ edges. There are several algorithms which give a polynomial time torus embedding routine (for example, [15] and [27]). The fastest of these is the linear time algorithm of Juvan, Marinček, and Mohar [22]. So far as we know, these have not been programmed and except possibly for [22], are solely of theoretical interest. An exponential algorithm was proposed by Neufeld and Myrvold [29], and computational experiments showed that it worked well in practice, at least for small graphs.

A *projective plane* is equivalent to a disk with antipodal points identified. A simple graph embeddable on the projective plane can have at most $3n-3$ edges. The first algorithm for embedding graphs in the projective plane was attempted by Perunicic and Duric [30]. Unfortunately, their algorithm is incorrect in that it sometimes fails to find a projective plane embedding of a graph when one exists as noted by Mohar [26, p. 483] and independently observed by Williamson (private communication to Mohar [26, p. 483]).

Mohar created a linear time algorithm for embedding graphs on the projective plane in 1993 [26]. This algorithm is fairly complex, and it would not be easy to program. The review of this paper in Mathematical reviews [37] states that "The process described is very complex (to program it would be a major undertaking)". Further, it states that "The reviewer has not had the time to check the numerous details needed to verify linearity". The new algorithm that we present in this paper is substantially simpler. However it is also slower, $O(n^2)$ instead of $O(n)$. If desired, the complexities of the linear time planarity algorithms can be avoided by using the much simpler algorithm of Klotz [23] for planar embedding and also finding Kuratowski subgraphs without compromising the $O(n^2)$ time bound.

Mohar's main motivation for building a fast projective plane embedding algorithm was the goal of finding a fast algorithm for checking embeddability of graphs in any surface. He uses a projective plane embedding algorithm together with a torus embedder as building blocks for the general case [27]. Clearly, a simple algorithm that is easy to program would help with these efforts.

Fiedler, Huneke, Richter, and Robertson have recently shown that the orientable genus of an arbitrary graph can be determined in polynomial time if the graph is projective planar [14]. This provides another practical application of a projective plane embedder.

Problems which are hard are often tractable if attention is restricted to graphs of fixed genus. For example, the problem of counting perfect matchings in a graph can be solved in polynomial time for graph of fixed orientable [16] or nonorientable [32] genus even though it is #P-complete in general [9]. To date, there has been little progress in generalizing techniques for planar graphs to fixed genus. Most such algorithms likely require an embedding of the graph. An algorithm for embedding that can easily be programmed opens the door to future research in this area that can be applied to practical applications.

Section 2 gives an introduction to orientable and nonorientable surface embeddings. Then the new algorithm is outlined in Section 3. The various components of the algorithm are then described in more detail in the sections that follow. The algorithm was programmed and used to computationally verify the set of projective plane obstructions [1]. The details are given in Section 8. The paper concludes with some ideas for future research.

2 Traversing faces

Algorithmic aspects for embedding on surfaces, especially nonorientable surfaces, are rarely addressed in texts on topology. To aid future work in this area, the critical concepts have been included in this section. For further information on combinatorial embeddings, Henle's *A Combinatorial Introduction to Topology* [20] provides an excellent introduction.

One way to draw a picture of a graph on an orientable surface having genus $g \geq 1$ is to start with a disk as pictured in Figure 2 [20, p. 109]. For each pair of arcs with the same label, an edge which exits through a point on one copy of the arc enters at the same place on the other copy of the arc as demonstrated with points u, v, and w in Figure 3.

To draw a graph of genus $g \geq 1$ on a nonorientable surface, a similar scheme is used. In this case, the arcs are as shown in Figure 4 [20, p. 112].

The orientable surfaces of genus 0 and 1 are called respectively a *plane*, and a *torus*. An orientable surface of genus $g \geq 2$ is called a *g-handled torus* because it can be obtained from the plane by adding g handles. The nonorientable surfaces of genus 1 and 2 are called respectively a *projective plane* and a *Klein bottle*.

The *orientable (nonorientable) genus* of a graph H is equal to the minimum g such that H can be embedded on an orientable (nonorientable) surface of genus g without edges crossing. The *orientable (nonorientable) genus* of an embedding $\tilde{\mathrm{H}}$ of a graph H is equal to the minimum g such that a picture of H which respects the clockwise order of $\tilde{\mathrm{H}}$ can be embedded on an orientable (nonorientable) surface of genus g without edges crossing.

An embedding of a graph on a surface can also be represented in a completely combinatorial fashion. One common scheme is to give a rotation system. A *rotation system* (also called a *combinatorial embedding*) is an adjacency list for a graph with the neighbours of each vertex cyclically ordered. If the surface is nonorientable, each edge also has an associated *signature* which is either $+1$ or -1. Intuitively, the cyclic adjacency list gives the clockwise order of the neighbours in the picture of the graph. The edges which go "over the boundary" have a -1 sign. The others have a $+1$ sign. Graphs embedded on an orientable surface can be treated uniformly by assigning a $+1$ signature to each edge.

There is a simple algorithm for walking around and recording each of the faces of such an embedding. The approach we present is a generalization of the standard approach for orientable surfaces. This approach is described by Archdeacon [2, p. 10] but extra detail is included here to make it easier

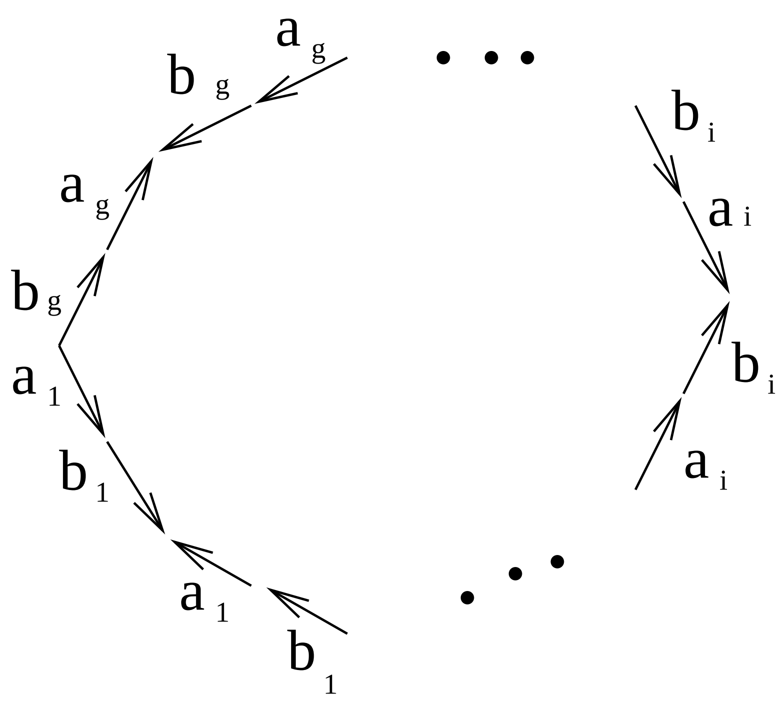


Figure 2: Representing a genus g orientable surface.

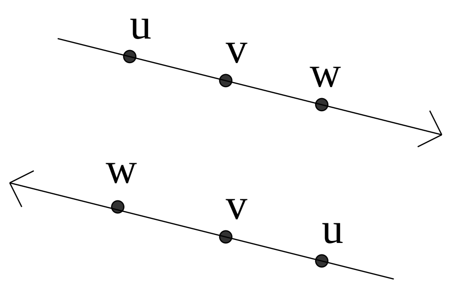


Figure 3: Corresponding points on two copies of an arc.

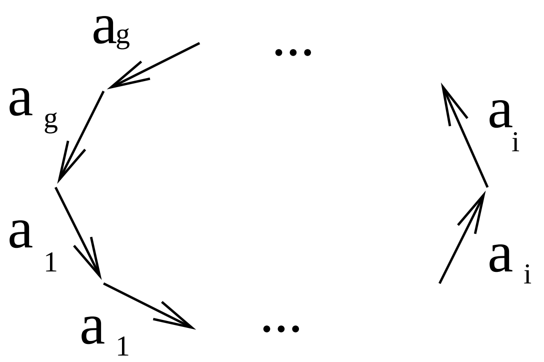


Figure 4: Representing a genus g nonorientable surface.

to implement.

Let $\Pi_b^1(a)$ be a function whose value is the neighbour which follows a in the cyclic list of neighbours for vertex b, and let $\Pi_b^{-1}(a)$ be the neighbour that precedes a. It is obvious from this definition that if $r = \pm 1$ and $c = \Pi_b^r(a)$ then $a = \Pi_b^{-r}(c)$. The signature of an edge e is denoted by $sgn(e)$.

Before describing the algorithm, the approach used to perform a facial walk is given. Each stage of a facial walk has a record which stores an arc (a, b) (directed edge) and a direction $r = \pm 1$ denoted by $[(a, b), r]$. The next record is $[(b, c), s]$ where

- $c = \Pi_b^r(a)$, and
- $s = r * sgn((b, c))$.

The walk terminates when $[(a, b), r]$ is revisited. The sequence of arcs traversed in the facial walk can be taken from the records where the last one (the repeated one) is ignored.

It is well known that in traversing the faces, each edge should be used two times. In an orientable surface, each edge is used once in each direction. For a nonorientable surface, the situation is more complex. If you traverse the faces using the picture of the graph, each edge is used twice, but some edges may be traversed twice in the same direction. This is formalized in the following theorem.

Theorem 2.1 *The walk starting at* $[(b, c), s]$ *visits vertices in the reverse order from the walk starting at* $[(c, b), -s * sgn((b, c))]$.

Proof. Consider a walk W_1 that goes from a to b to c with $[(a, b), r]$ followed by $[(b, c), s]$. The reverse walk W_2 should go from c to b to a. Assume the records used are $[(c,b),t]$ followed by $[(b,a),u]$. The first step of W_1 implies that $c = \Pi_b^r(a)$. Further, it is obvious from the definition of $\Pi_b^r(a)$ that if $c = \Pi_b^r(a)$ then $a = \Pi_b^{-r}(c)$. Thus in the reverse walk W_2, $t = -r$. The theorem states that $-s * sgn((b,c))$ should equal $t = -r$. From the definition of a facial walk, $s = r * sgn((b,c))$. So $-s * sgn((b,c)) = -r * sgn((b,c))^2$. Because the value of $sgn(b,c)$ is either $+1$ or -1, $sgn((b,c))^2 = 1$. Hence $-s * sgn((b,c)) = -r = t$ as required. ∎

The proof of the above theorem demonstrates that just as there is only one way to step forwards in a facial walk, from a given record there is only one way to select its predecessor. From this, it is easy to prove that the starting record must be the first record repeated in a facial walk. This is obvious from a topological point of view, but it is also possible to give a combinatorial proof.

Theorem 2.2 *Every facial walk eventually terminates, and at the end, the only record that has been repeated is the starting record.*

Proof. Because there are only a finite number of possibilities for the records, eventually one must be repeated. Suppose that in the walk, the first time a repeated record is encountered is with $[(a,b),r]$. If this is not the first record of the walk, then the record which occurs before it the first time must also occur before it the second time. This contradicts the assumption that $[(a,b),r]$ is the first repeated record. ∎

Because the walk starting at $[(b,c),s]$ is equivalent to the one starting at $[(c,b),-s * sgn((b,c))]$ only one of these should be used in the process of traversing all the faces. To traverse all the faces, the records considered at the start are $[(b,c),+1]$ and $[(b,c),-1]$ for each edge (b,c) where $b < c$. If a record $[(c,b),s]$ is encountered having $c > b$, it is considered to be equivalent to $[(b,c),-s * sgn((b,c))]$.

The algorithm for traversing all the faces can now be described. There are two records for each edge $(b,c) : [(b,c),+1]$ and $[(b,c),-1]$ where without loss of generality, $b \leq c$. First initialize all records to be unvisited. Then, while unvisited records remain, pick an unused record and traverse the face that it contains. As the face is traversed, each record encountered is marked as visited.

If a doubly linked cyclic adjacency list is used, then each step of a walk takes only constant time. In traversing all the faces, the number of steps is twice the number of edges of the graph. Hence, it is possible to traverse all the faces in $O(m)$ time.

It is well known that any face of an embedding of a nonorientable graph must contain an even number of edges which go over the boundary. Again, a simple combinatorial proof is possible.

Theorem 2.3 *Any face of an embedding contains an even number of edges with -1 sign.*

Proof. Suppose the facial walk starts with the record $[(a, b), r]$. For each subsequent record of the walk, the direction term is multiplied by the signature of the edge that is used. Thus to return to $[(a, b), r]$, the number of edges in the walk with -1 signature must be even. ∎

Any combinatorial embedding that has only $+1$ signatures on the edges is an *orientable embedding*, otherwise, it is a *nonorientable embedding*. The *genus* of a orientable embedding (nonorientable embedding) is the smallest g such that the embedding is realizable with g handles (g crosscaps). The *genus* of an embedding of a connected graph can also be defined combinatorially by the following formula.

Theorem 2.4 *[20] The genus g of the embedding $\tilde{H}$ of a connected graph H equals*

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

for an orientable surface and

$$2 - n + m - f$$

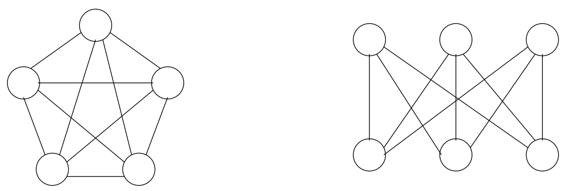
for a nonorientable surface where n is the number of vertices, m is the number of edges, and f is the number of faces. ∎

The *orientable (nonorientable) genus* of a graph H is equal to the minimum g such that H has an orientable (nonorientable) combinatorial embedding of genus g. The formula given above together with the observation that each face has at least three edges gives the well known upper bounds for the maximum number of edges of a simple graph embedded on a suface of genus g. For an orientable surface with g handles, a simple graph which is embeddable on this surface has at most $3n - 6 + 6g$ edges and for a nonorientable surface with g crosscaps, the maximum is $3n - 6 + 3g$. Hence the maximum number of edges is $3n - 6$ for a plane, $3n$ for a torus, $3n - 3$ for a projective plane and $3n$ for a Klein bottle.

The output of an embedding algorithm is a combinatorial embedding (with signs associated with the edges for nonorientable surfaces). It is easy to check that the embedding has the required genus using the face walking algorithm described in this section, together with Theorem 2.4.

3 The basic idea

Both Mohar's algorithm [26] and our own take a similar approach to embedding graphs in the projective plane. If a graph is planar, it is also



K_5 $\quad$ $K_{3,3}$

Figure 5: The planar obstructions K_5 and $K_{3,3}$

projective planar, and a planar embedding also constitutes an embedding in the projective plane. A subgraph is *homeomorphic* to K_5 (or $K_{3,3}$) if it is as pictured as in Figure 5, except that each edge could possibly be replaced by a path. If a graph is not planar, the linear time planarity algorithms can be modified (see for example [36]) to find a subgraph homeomorphic to $K_{3,3}$ or K_5 which we call K. An alternative approach which avoids the complexities of the linear time planarity testers without increasing the running time of our algorithm is to apply the simpler $O(n^2)$ algorithm of Klotz [23].

For each way of embedding K on the projective plane, the basic idea is to try and embed the rest of the graph in the faces of the embedding of K. Observe that if just one face is considered at a time, the embedding is locally planar (see Figures 6 and 7). This section describes the different ways to embed $K_{3,3}$ or K_5 in the projective plane (taken from Mohar [26]).

We define two embeddings of K in the projective plane to be *equivalent* if their faces are the same (ignoring any sense of clockwise as this has no meaning on a nonorientable surface). The following theorem describes the non-equivalent embeddings of $K_{3,3}$ and K_5. These ways of labellings the embeddings were hard coded into our computer program.

Theorem 3.1 *[26] The unlabelled embeddings of $K_{3,3}$ in the projective plane are as drawn in Figure 6 and those of K_5 are as pictured in Figure 7 (a) and (b). Further, the number of non-equivalent ways to label these is six for $K_{3,3}$, twelve for K_5 as shown in Figure 7(a), and fifteen for K_5 as shown in Figure 7(b).* ∎

A *bridge* of a graph G with respect to an embedded subgraph H is a subgraph of G which is either (1) an edge not in H whose endpoints are both in H plus its endpoints or (2) a connected component of $G - H$ together with the edges which connect a vertex in the connected component to a vertex in H and their endpoints. The vertices that a bridge shares with H are called its *attachment points*.

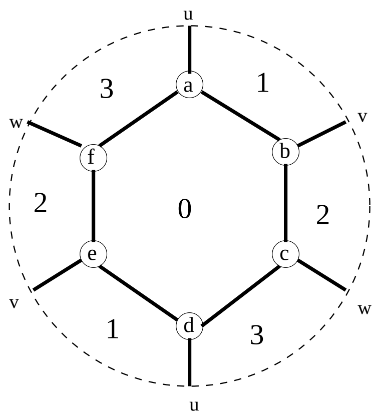


Figure 6: $K_{3,3}$ embedded in the projective plane

A bridge can be *drawn* in a face F if its points of attachment all lie on F. A bridge B can be *embedded* in a face F if there is a planar embedding of $B \cup F$. A k-face bridge with respect to an embedded subgraph $\tilde{K}$ is a bridge that can be embedded in k faces. Starting with a 2-vertex connected graph, all bridges for $\tilde{K}$ where $\tilde{K}$ is an embedding of K_5 or $K_{3,3}$ in the projective plane are either 1-face, 2-face or 3-face bridges.

The major difference between our algorithm and Mohar's is in the process used to embed the rest of the graph. The new algorithm considers various assignments of the 3-face bridges to faces (Section 4). After a preliminary determination of bridges which cannot embed in each face together (Section 5), an algorithm for 2-SAT is used to determine an assignment of bridges to faces that can be extended to an embedding of the graph if one exists. This process is described in Section 6. Finally, the time complexity of the new algorithm is analyzed in Section 7. Section 9 suggests some areas for future research.

4 The 3-face bridges

Assume that a Kuratowski subgraph K is embedded in one of the ways listed in Theorem 3.1. Even when the input graph is 2-vertex connected, there can be bridges of the graph which are embeddable in three faces. This section describes the possibilities for 3-face bridges and explains how these are handled by our new algorithm.

The possible sets of attachments for the 3-face bridges are:

For $\mathbf{K_{3,3}}$ as pictured in Figure 6 :

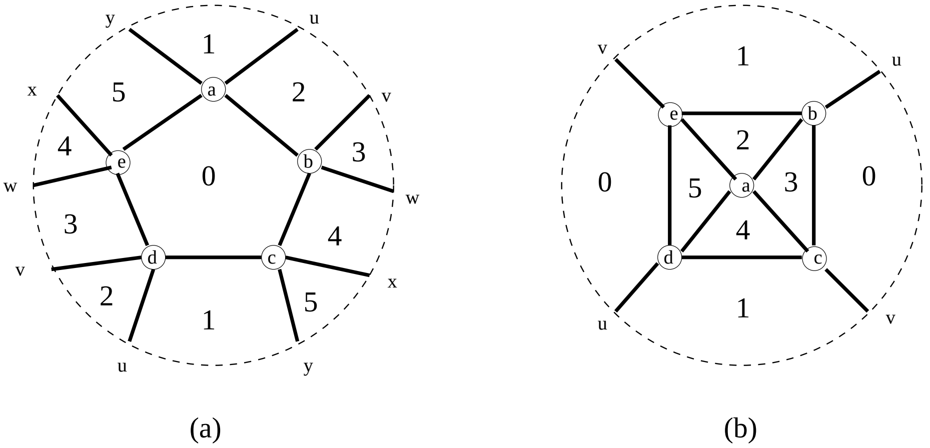


Figure 7: K_5 embedded in the projective plane

- $\{a, d\}$ (faces: 0, 1, 3),
- $\{b, e\}$ (faces: 0, 1, 2), and
- $\{c, f\}$ (faces: 0, 2, 3).

For $\mathbf{K_5}$ as pictured in Figure 7(a) :

- $\{a, c\}$ (faces: 0, 1, 5),
- $\{a, d\}$ (faces: 0, 1, 2),
- $\{b, d\}$ (faces: 0, 2, 3),
- $\{b, e\}$ (faces: 0, 3, 4), and
- $\{c, e\}$ (faces: 0, 4, 5).

For $\mathbf{K_5}$ as pictured in Figure 7(b) :

- $\{b, c\}$ (faces: 0, 1, 3),
- $\{b, e\}$ (faces: 0, 1, 2),
- $\{c, d\}$ (faces: 0, 1, 4), and
- $\{d, e\}$ (faces: 0, 1, 5).

The 3-face bridges with the same sets of attachment points can be treated as a unit. Thus, there is only some constant number of ways that the 3-face bridges can be assigned to faces. Our aim is to choose a selection of these cases which covers all possibilities, and leaves at most two faces as options for each bridge (this is required to apply 2-SAT as per Section 6).

The 3-face bridges are assigned to a *type* which corresponds to their attachment points. For $K_{3,3}$ as pictured in Figure 6, observe that at most one type of 3-face bridge can be embedded into the face numbered 0 at one time. Thus, there are four choices for placement of the 3-face bridges- the fourth choice corresponding to embedding none of the 3-face bridges in face 0.

We now do the same type of analysis for the embeddings of K_5 as pictured in Figure 7(a). Note that for all 3-face bridges, one of the possible face choices is face 0. Thus, it is sufficient to choose the bridges which are assigned to face 0 as this leaves only two choices for the bridges which are not. There are five ways to assign two 3-face bridges into face 0 (the pair must have a common attachment point), five ways to assign one to this face, and one choice which has no 3-face bridges in face 0. Thus, in total there are eleven possibilities.

For K_5 as pictured in Figure 7(b), observe that bridges with attachment sets $\{b, e\}$ and $\{c, d\}$ cannot both be embedded into face 0 at the same time. Similarly, bridges with attachment sets $\{b, d\}$ and $\{c, e\}$ cannot both be embedded into face 1 at the same time. There are nine ways to assign bridges to faces so that face 0 contains either one or none of the bridge types $\{b, e\}$ and $\{c, d\}$, and face 1 contains either one or none of the bridge types $\{b, d\}$ and $\{c, e\}$. Further, these leave at most two choices for a face for each bridge that is not assigned. Thus, only these nine possibilities need to be considered.

In summary, the number of types of 3-face bridges can be at most three for $K_{3,3}$, five for K_5 as drawn in Figure 7(a), and four for K_5 as drawn in Figure 7(b). The number of assignments of these to faces that are considered is four for $K_{3,3}$, eleven for K_5 as drawn in Figure 7(a), and nine for K_5 as drawn in Figure 7(b).

5 Computing conflicts between bridges

Two bridges are *compatible* for a face if both can be embedded inside the face simultaneously. Otherwise, they are said to be *conflicting*. For the plane, it is a well known fact that if a set of bridges are pairwise compatible for a face, then they can all be embedded in the face simultaneously. This applies here because the faces are homeomorphic to a face on the plane. This section describes how the new algorithm determines the conflicting pairs of bridges.

There can be a linear number of bridges, and hence the number of bridge pairs can be quadratic. In order for the entire algorithm to run in $O(n^2)$ time, information about bridge conflicts must be computed in $O(n^2)$ total time. We now explain how this can be accomplished.

The vertices of the face are processed in cyclic order. A finite state machine (FSM) is used for each pair of bridges (B_i, B_j) to determine whether they conflict. Using finite state machines enables us to determine all the bridge conflicts "in parallel" as we are traversing the face, and this is critical for obtaining the $O(n^2)$ time complexity for this algorithm. The FSM makes its transitions depending on whether the face vertex being processed is an attachment of B_i, B_j, or both. If it is an attachment to neither, nothing happens to the FSM for (B_i, B_j) at this stage.

The states of this FSM are:

Start: the start state.

Seen_i: have seen attachments to B_i but not to B_j.

Seen_j: have seen attachments to B_j but not to B_i.

Seen_shared: initially there is a shared attachment point.

Seen_shared2: first two attachment points are both shared.

Seen_i_then_j: have seen an attachment for B_i, now examining attachments for B_j.

Seen_j_then_i: have seen an attachment for B_j, now examining attachments for B_i.

Only_i: only attachments to B_i are allowed.

Only_j: only attachments to B_j are allowed.

Conflicting: the bridges conflict.

If the FSM terminates in the state *Conflicting*, then the bridges conflict. Otherwise, they do not. The transitions are shown in Table 1.

There can be on the order of n cycle vertices that must be processed. For each cycle vertex v, the number of FSM's updated (in $O(1)$ time each) is on the order of $O(na * nb)$ where na is the number of bridges attaching at v, and nb is the number of bridges embeddable in the face. So the total work is $O(nt * nb + n)$ where nt is the total number of attachments to the face. This is in $O(n^2)$ because a graph having at most $3n - 3$ edges can have no more than $O(n)$ attachments or bridges.

6 Using 2-SAT to assign bridges to faces

Our aim is to assign bridges to faces of the embedded Kuratowski subgraph in order that the input graph can be embedded in the projective plane. In

Current state	Next state if face vertex attaches to:		
	B_i but not B_j	B_j but not B_i	B_i and B_j
Start	Seen_i	Seen_j	Seen_shared
Seen_i	Seen_i	Seen_i_then_j	Seen_i_then_j
Seen_j	Seen_j_then_i	Seen_j	Seen_j_then_i
Seen_shared	Seen_j_then_i	Seen_i_then_j	Seen_shared2
Seen_shared2	Only_i	Only_j	Conflicting
Seen_i_then_j	Only_i	Seen_i_then_j	Only_i
Seen_j_then_i	Seen_j_then_i	Only_j	Only_j
Only_i	Only_i	Conflicting	Conflicting
Only_j	Conflicting	Only_j	Conflicting
Conflicting	Conflicting	Conflicting	Conflicting

Table 1: Transitions of the FSM for determining bridge conflicts.

the special case that each bridge can be assigned to at most two faces, a 2-SAT system can be created that either provides such an assignment of bridges to faces, or indicates that it is impossible as described below.

A literal is a variable, say x or its complement $\bar{x}$. Clauses are sets of literals. The satisfiability problem (SAT) is to determine whether there is an assignment of *true/false* values to the variables so that at least one literal in each clause evaluates to *true*.

It is well-known that SAT is NP-complete [10], even in the case that each clause contains at most three variables (3-SAT) [17, pp. 46-50]. However, if each clause has at most two variables (2-SAT), the problem has a simple algorithm that takes time which is linear in the input size [13].

Let f be a face of $\tilde{K}$ and b be a bridge of $\tilde{K}$ that can be embedded in f. The literal (b, f) if assigned the value *true* indicates that bridge b is assigned to face f. If a bridge b can only be assigned to face f, then the clause $\{(b, f), (b, f)\}$ is added to the 2-SAT system (or equivalently, you could just set the value of (b, f) to *true*). For a bridge b that is embeddable in faces f_1 and f_2, the two clauses $\{(b, f_1), (b, f_2)\}$ and $\{\overline{(b, f_1)}, \overline{(b, f_2)}\}$ together ensure that b is assigned to exactly one of these faces. For each pair of conflicting bridges b_1 and b_2 for a face f, the clause $\{\overline{(b_1, f)}, \overline{(b_2, f)}\}$ ensures that they are not assigned to f simultaneously.

There can be at most $O(n)$ bridges, and at most $O(n^2)$ conflicts between pairs of bridges. Thus, the algorithm for 2-SAT [13] takes $O(n^2)$ time to determine an assignment of bridges to faces if one exists.

7 Our algorithm and its time complexity

This section first gives a high level description of our new algorithm. This is followed by an analysis of the time complexity.

The first step of our algorithm is to find the maximal 2-vertex connected subgraphs (*blocks*) of the input graph as is standard in any embedding algorithm. This can be done in $O(n)$ time using a modified depth first search (see for example [18]). It is well-known that if all the blocks are planar except one which could be projective planar, then G is projective planar.

Thus, the pseudocode below is for a simple graph G which has no cut vertices. It either computes a projective planar embedding of G or returns *false* to indicate that no such embedding exists.

Projective plane embedding algorithm
(1) If $m > 3n - 3$ return *false.*
(2) If G is planar, return a planar embedding of G.
(3) Find a subgraph K homeomorphic to $K_{3,3}$ or K_5.
(4) For each labelled projective planar embedding $\tilde{K}$ of K do:
(5)   Find all the bridges of $\tilde{K}$ and determine which faces they can be embedded in.
(6)   If a bridge b cannot be embedded in any face of $\tilde{K}$ go to (12).
(7)   Compute the conflicts between pairs of bridges.
(8)   For each arrangement of 3-face bridges do:
(9)   If there is an assignment of bridges to the faces of $\tilde{K}$,
(10)   return a projective planar embedding.
(11)   End for (8).
(12) End for (4).
(13) Return *false.*

If there are more than $3n-3$ edges, the input graph cannot be projective planar, and the algorithm terminates at Step 1. For the remaining steps, the number of edges is at most $3n - 3$ which is in $O(n)$ and this is critical in the analysis.

If the graph G is planar, it is also projective planar. This is detected at Step 2 using any of the $O(n)$ planarity testing algorithms (for example, [7]). If the graph is not planar, a Kuratowski subgraph K can be extracted in $O(n)$ time (see for example [36]). Klotz [23] provides a much simpler approach that runs in $O(n^2)$ time.

The loop at Step 4 is executed at most six times if K is homeomorphic to $K_{3,3}$ and 27 times for K_5 (Theorem 3.1). In either case, there is only a constant number of iterations.

At Step 5, the bridges can be found in $O(n)$ time using a modified BFS algorithm. Determining the faces that each bridge can be embedded in (Steps 5 and 6) can be accomplished using a planarity tester. If the graph tested for planarity is the bridge with its points of attachment connected in a cycle which respects the cyclic order of the face, the total time for all the tests is in $O(n)$ using a linear time planarity tester. The number of faces considered is four for $K_{3,3}$ or six for K_5. Thus, the total work at this step is $O(n)$.

The method for computing conflicts between pairs of bridges (Step 7) is described in Section 5. As noted, the time is in $O(n^2)$.

For Step 8, the number of arrangements of the 3-face bridges is a constant as indicated in Section 4. Step 9 is solved using a procedure for 2-SAT as described in Section 6. As noted earlier, the time taken is in $O(n^2)$.

In Step 10, the projective planar embedding can be described by indicating a planar embedding for the contents of each of the faces. To embed a collection of bridges inside a face f, connect a new vertex w to each vertex of the face. Add the bridges and then invoke a standard planarity tester. Without loss of generality, assume w is embedded outside the face. Any bridges embedded in the outside of the face can attach only at two adjacent points on the face. These bridges can be easily flipped to the inside of the face to obtain an embedding of the face with all the bridges on the inside. Thus, it takes $O(n)$ time in total to find the projective planar embedding once the bridges are assigned appropriately to faces.

Because each of the loops in the above code only involves a constant number of iterations, the time for the whole algorithm is in $O(n^2)$. To improve it to linear time, it would be necessary to speed up Steps 7 and 9. Note that the complexities of the linear time planar embedding algorithms can be completely avoided while retaining the overall $O(n^2)$ time complexity by using one of the straightforward $O(n^2)$ planar embedding algorithms [12, 23]. Klotz [23] also provides the required Kuratowski subgraph in $O(n^2)$ time. Unlike many of the algorithms of theoretical interest, the constant overhead in this algorithm is very reasonable.

8 Computational Results

We programmed a variant of this algorithm. Simplicity, correctness, and speed of implementation were more important to us than the execution speed of the final code. For this reason, the simple $O(n^2)$ planarity algorithm of Demoucron, Malgrange, and Pertuiset [12] was used for planar embedding. A Kuratowski subgraph was isolated by a simple $O(n^3)$ approach:

Input: a non-planar graph G.

Output: a subgraph of K homeomorphic to either K_5 or $K_{3,3}$.

Set K to be equal to G.

For each edge e of K do:
  If $K - e$ is planar, keep e in K,
  otherwise, delete e from K.

The algorithm of Klotz [23] provides a simple $O(n^2)$ alternative. We chose the Democron algorithm instead because we already had an implementation of it which has been thoroughly debugged and tested in conjunction with a related project whose aim was to find torus embeddings [29].

A *topological obstruction* for a surface M is a graph G with no vertices of degree less than three such that G is not embeddable on M, but $G - e$ is embeddable on M for all edges e of G. A *minor order obstruction* has the additional property that G with edge e contracted is embeddable on M for all edges e. For any surface of fixed genus, Robertson and Seymour theory indicates that the number of obstructions (topological or minor order) is finite [31]. A proof specifically for orientable surfaces has been provided by Bodendiek and Wagner [5] and for nonorientable surfaces by Archdeacon and Huneke [3].

It is well-known that the topological obstructions [24] and the minor order obstructions [34] are the same for the plane. These graphs are K_5 and $K_{3,3}$ (pictured in Figure 5).

For the projective plane, there are 103 topological obstructions and 35 minor order obstructions. Glover, Huneke and Chin first proved that these were obstructions [19]. Archdeacon later proved that their list was complete [1]. We verified this result computationally by checking all graphs on up to 11 vertices with up to $3n + 1$ edges (generated using McKay's *makeg* program based on his program *nauty* [25]). We found exactly the same obstructions. These computations provide added confidence that both our program, and the theoretical result of Archdeacon [1] are correct.

It would have sufficed to test graphs on up to $3n - 2$ vertices because a projective planar graph has at most $3n - 3$ edges so an obstruction has at most $3n - 2$. The extra computation resulted from not immediately realizing that Mohar's upper bound of $3n$ for the number of edges in a projective planar graph [26, p. 484] was not tight and that Euler's formula actually gives an upper bound of $3n - 3$ and not $3n$.

9 Future research

A similar approach could be attempted for designing an algorithm for embedding graphs on the torus. Unfortunately, things are not as simple for the torus because there are embeddings of $K_{3,3}$ and K_5 whose faces include repeated vertices. A fast torus embedding algorithm would be a useful tool for computing all the obstructions for the torus. The ones known to date are summarized in [29] and more recently [8]. So far, it is known that there are at least 239,322 topological obstructions and 16,629 minor order obstructions [8].

Finding the genus of an arbitrary graph is NP-complete [33]. While development of a polynomial time algorithm is not likely, an exponential algorithm whose performance was reasonably fast in practice would be a useful tool. Such algorithms have been developed for torus embedding [29] and for various NP-complete problems such as CLIQUE (see for example [28]).

Finally, many problems that can be solved on the plane in polynomial time in spite of being intractable in general likely also have polynomial time algorithms for surfaces of fixed genus. The development of practical embedding algorithms motivates the search for these types of algorithms. To date, most of the work has been focussed on orientable surfaces, but it is likely that these kinds of algorithms also exist for nonorientable surfaces such as the projective plane.

References

[1] D. Archdeacon. The complexity of the graph embedding problem. In (R. Bodendiek and R. Henn), editors, *Topics in combinatorics and graph theory*, pages 59–64, Heidelberg, 1990. Physica-Verlag.

[2] D. Archdeacon. Topological graph theory: A survey. *Cong. Num.*, 115:5–54, 1996.

[3] D. Archdeacon and J. P. Huneke. A Kuratowski theorem for nonorientable surfaces. *JCT B*, 46:173–231, 1989.

[4] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Upper Saddle River, NJ, 1999.

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

[6] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ–tree algorithms. *J. Comput. Sys. Sci.*, 13:335–379, 1976.

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

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

[9] C. Colbourn, J. S. Provan, and D. Vertigan. The complexity of computing the Tutte polynomial of transversal matroids. *Combinatorica*, 15(1):1–10, 1995.

[10] S. A. Cook. The complexity of theorem-proving procedures. In *3rd ACM Symp. Theory of Comp.*, pages 151–158, 1971.

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

[12] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires. *Rev. Française Recherche Opérationnelle*, 8:33–47, 1964.

[13] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. of Computing*, 5:691–703, 1976.

[14] J. R. Fiedler, J. P. Huneke, R. B. Richter, and N. Robertson. Computing the orientable genus of projective graphs. *J. Graph Theory*, 20(3):297–308, 1995.

[15] I. S. Filotti, G. L. Miller, and J. Reif. On determining the genus of a graph in $O(v^{O(g)})$ steps. In *11th ACM Symp. Theory of Comp.*, pages 27–37, 1979.

[16] A. Galluccio and M. Loebl. On the theory of Pfaffian orientations, I: Perfect matchings and permanents. *Electron. J. of Combin.*, 6:1–18, 1999.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[18] A. Gibbons. *Algorithmic graph theory.* Cambridge University Press, New York, 1985.

[19] H. H. Glover, J. P. Huneke, and C. S. Wang. 103 graphs that are irreducible for the projective plane. *J. Combin. Theory*, 27(3):332–370, 1979.

[20] Michael Henle. *A combinatorial introduction to topology.* Dover publications, Inc., New York, 1994.

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

[22] M. Juvan, J. Marinček, and B. Mohar. Embedding a graph in the torus in linear time. In *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, Vol. 920*, pages 360–363. Springer, Berlin, 1995.

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

[24] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930.

[25] B. D. McKay. Practical graph isomorphism. *Proc. Tenth Manitoba Conf. Numerical Math. and Computing, Cong. Num.*, 30:45–87, 1981.

[26] B. Mohar. Projective planarity in linear time. *J. Algorithms*, 15:482–502, 1993.

[27] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.*, 12(1):6–26, 1999.

[28] W. Myrvold, T. Prsa, and N. Walker. A dynamic programming approach for timing and designing clique algorithms. *Algorithms and Experiments (ALEX '98): Building Bridges Between Theory and Applications*, pages 88–95, 1998.

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

[30] B. Perunicic and Z. Duric. An efficient algorithm for embedding graphs in the projective plane. In *The fifth quadrennial international conference on the theory and applications of graphs with special emphasis on algorithms and computer science applications*, pages 637–650, Western Michigan University, June 4-8, 1984. John Wiley and Sons, Inc.

[31] N. Robertson and P. Seymour. Graph minors - a survey. In *Surveys in Combinatorics.* Cambridge University Press, 1985.

[32] G. Tesler. Matchings in graphs on non-orientable surfaces. *JCT B*, 78:198–231, 2000.

[33] C. Thomassen. The graph genus problem is NP-complete. *J. Algorithms*, 10:568–576, 1989.

[34] K. Wagner. Über einer eigenschaft der ebener complexe. *Math. Ann.*, 114:570–590, 1937.

[35] S. G. Williamson. Embedding graphs in the plane- algorithmic aspects. *Ann. Disc. Math.*, 6:349–384, 1980.

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

[37] S. G. Williamson. *Math. Reviews*, 94f:05141, 1994.