Practical Toroidality Testing

by

Eugene T. Neufeld

BA, University of B.C., Vancouver, Canada, 1963

[redacted]

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of
Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

Dr. Wendy Myrvold, Supervisor (Department of Computer Science)

[redacted]

Dr. Frank Ruskey, Departmental Member (Department of Computer Science)

[redacted]

Dr. William Pfaffenberger, Outside Member (Department of Mathematics and Statistics)

[redacted]

Dr. Luis Goddyn, External Examiner (Department of Mathematics and Statistics)

Supervisor: Dr. Wendy Myrvold

Abstract

A torus is a sphere with one handle. A toroidal graph is one that can be embedded in the torus with no overlapping edges. Although various theoretically efficient algorithms for testing toroidality of graphs have been published, none is claimed to be "practical". We develop an exponential toroidality tester that is practical enough to compute the minor order obstructions to toroidality on up to ten vertices and which gives indications that there are several thousand of these obstructions all together. We validate the algorithm combinatorially, and discuss prospects of improvements in efficiency.

Examiners:

Dr. Wendy Myrvold, Supervisor (Department of Computer Science)

Dr. Frank Ruskey, Departmental Member (Department of Computer Science)

Dr. William Plaffenberger, Outside Member (Department of Mathematics and Statistics)

Dr. Luis Goddyn, External Examiner (Department of Mathematics and Statistics)

Contents

List of Figures

Acknowledgements

I acknowledge from my supervisor, Dr. Wendy Myrvold, Research Assistantship support, a very nice problem, copious critical feedback, improvements in various subroutines, and algorithm $\mathcal{C}$ and its initial implementation, including the idea of partitioning edges and modifying Horton's minimum weight cycle basis algorithm to find a basis of cycles with a minimum number of incident edges. The germinal idea of removing a noncontractible cycle and considering the remaining planar graph is Mr. Jim Wall's. I owe Dr. William Pfaffenberger, my very gracious and inspiring mathematics teacher, the encouragement to persist with the best effort at critical moments.

Financial assistance from my parents and from the student loan programs of B.C. and of Canada are also gratefully acknowledged.

Thank you, Fiona, for your faithful confidence.

Chapter 1

Introduction

A *graph*, G, the object in whose toroidality we are interested, is defined to be a finite set $V(G)$ of vertices, together with a subset $E(G)$ of pairs of distinct elements of $V(G)$ which we call *edges* of G. Roughly speaking, identifying edges with line segments between vertices, a graph G *embeds* in a two dimensional surface S if G maps into S without overlapping vertices or intersecting edges. A graph which embeds in the sphere (equivalently, plane) is called *planar*, and a graph which embeds in the torus (that is, the sphere with one handle—or, equivalently, one hole) is called *toroidal.* According to our definition, graphs cannot have loops or multiple edges. These objects are often called *simple graphs* in the literature. But the question whether a graph with multiple edges and loops embeds in a given surface reduces immediately to the question for a simple graph, so our only interest here is in simple graphs. We simply call them *graphs.*

Although practical, theoretically efficient planarity testers abound, and although very simple, as well as theoretically efficient toroidality testers are known, there seem to be no practical toroidality testers available. We develop and implement a toroidal-

ity tester that is practical enough to compute the minor order obstructions (see definition in Chapter 6, Section 2) to toroidality on up to ten vertices, as well as to indicate that there exist in all at least several thousand of these.

In the first section we introduce some terms. In Section 2 we state our problem and give a history of the problem and its relatives. In Section 3 we present some motivation for an interest in graph genus and in practical solutions to the problems. In Section 4 we sketch an algorithm (labelled $\mathcal{B}$) based on an idea of Jim Wall, which we justify combinatorially in Chapter 2, and refine in Chapter 3 to algorithm $\mathcal{C}$. The main contributions of this thesis, outlined in the final section (5) of this introduction, are the validation in rigorous combinatorial terms, of algorithms $\mathcal{B}$ and $\mathcal{C}$, the development, combinatorial validation, and implementation of algorithms $\mathcal{D}$ and $\mathcal{E}$ presented in Chapters 4 and 5, and the tables of numbers of obstructions of various types presented in Chapter 6. We show incidentally that there are substantially more minor order obstructions to toroidality than might have been hoped, based on previously existing results.

1.1 Definitions

The definitions, propositions, and style of presentation owe most to Thomassen [Tho90]. For proofs of elementary graph theoretic propositions throughout, we refer to Bondy and Murty [BM76] and to Gibbons [Gib85].

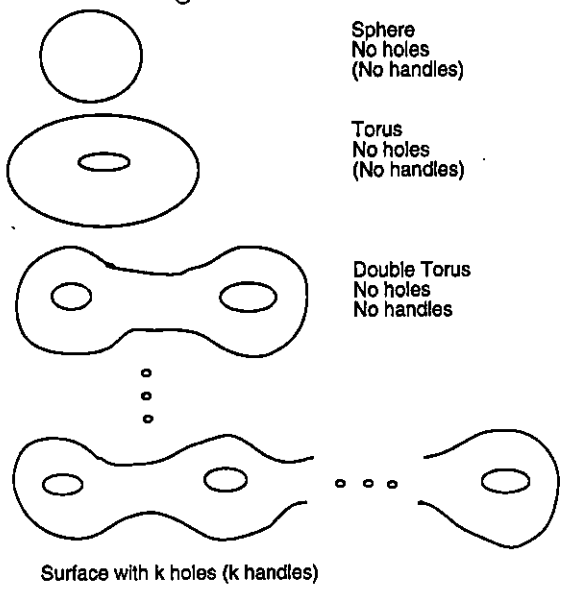
A *subgraph* H of graph G is a graph with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We use the convention that n_v and n_e refer respectively to the number of vertices, $|V(G)|$, and the number of edges, $|E(G)|$, of G. For the sake of efficient data structures, vertices are labelled 0 to $n_v - 1$. We say vertices v and w are adjacent when $\{v, w\} \in E(G)$. A graph can be specified or presented as a system of *adjacency lists*, one list

Figure 1.1: $K_{3,3}$

Adjacency Lists:	Faces:
0: 3,4,5	0,3,1,4,2,5,0
1: 3,4,5	0,4,1,5,2,3,0
2: 3,4,5	0,5,1,3,2,4,0
3: 0,1,2	
4: 0,1,2	
5: 0,1,2	
(a)	(b)

per vertex. The adjacency list for a vertex v of G lists the vertices of G adjacent to v. Figure 1.1 (a) shows an adjacency list representation of the graph commonly named $K_{3,3}$. An adjacency list is not considered as ordered. The surfaces in which we embed may be considered to be the two dimensional compact oriented surfaces without boundaries in Euclidean three-space: the sphere, the torus, the double torus, and in general, the sphere with k "handles", which is said to have *genus* k. (See Figure 1.2 for diagrams of these surfaces.) In fact, however, the objects, proofs, and algorithms we present here, starting in Chapter 2, are combinatorial rather than topological. Thus, by an *embedding* of a graph G we really mean a presentation of G as a system of *cyclically ordered* adjacency lists (also called a *rotation system*) for the vertices of G. There are two possible *orientations* for each list, and this orientation matters. Inverting (the orientation of) a single list in an embedding gives a different embedding. But it is only the *relative* orientation that matters. That is, if every list of an embedding Π is inverted, (we call this a *reflection* of the embedding) the resulting

Figure 1.2: Surfaces



embedding is considered to be identically Π. Our representation of a graph G is therefore exactly the same as our representation of a particular embedding of G: the difference lies in our interpretation of the adjacency lists as being cyclically ordered or simply sets. Topologically, an embedding of G in a surface S is a well behaved mapping of the graph considered as a set of vertices and line segments, into S. This representation is related to the combinatorial one we employ by considering, in the topological embedding, at any particular vertex v, the clockwise circular sequence of vertices adjacent to v (or, seen from the other side of the surface, the counter-clockwise sequence).

In the topological view, a graph is seen to partition the surface in which it is embedded into "faces". Combinatorially we define faces as follows. A *walk* in G is a finite sequence $v_1, v_2, ..., v_p$ in $V(G)$ with $(v_i, v_{i+1}) \in E(G)$ for $1 \leq i \leq p-1$. A *closed walk* is a walk in which $v_1 = v_p$. A *path* is a walk with no repeated vertices. A *cycle* is a walk with no repeated vertices except $v_1 = v_p$. Closed walks and cycles

may be represented by making any of their vertices first. That is, the representing sequences are cyclic. Though cycles, like edges do not have orientation, we sometimes consider them together with an orientation. Edge $\{a, b\}$ may be oriented as arc (a, b) or arc (b, a). All cycles are represented using parentheses, so oriented cycles will be explicitly referred to as such.

Before defining face and genus we need a little more notation for embeddings. If the cyclically ordered list (of adjacent vertices) of a vertex v of embedding Π of graph G is $\Pi_v = u_1, u_2, \ldots, u_d$, then using the notation of permutation theory, in which the cyclically ordered list would be called a circular permutation, we write $u_d = u_0$ and $\Pi_v^k(u_j) = u_{(j+k) \bmod d}$. We write $\Pi_v(x)$ for $\Pi_v^1(x)$. For example $\Pi_v(u_1) = u_2$, $\Pi_v^2(u_1) = u_3$, $\Pi_v^2(u_d) = u_2$ and $\Pi_v^d(u) = u$ for any u in the adjacency list of v.

We define a *face* of an embedding Π to be a minimal closed walk such that for $e = (v_i, v_{i+1})$ in the walk, $\Pi_{v_i}(v_{i-1}) = v_{i+1}$ for all i. Thus faces may have repeated vertices, and even repeated edges (when an edge is "traversed" once each way). In fact we observe that each edge occurs exactly twice in the set of faces of an embedding: each edge is "traversed" once each way. We may designate the face determined by arc (a, b) as $Face(\Pi, (a, b))$. $Face(\Pi, (a, b)) = (a, b, \Pi_b(a), \Pi_{\Pi_b(a)}(b), \ldots, a)$. Examples of faces for different embeddings of the same graph can be seen in Figures 1.1 (b) and 1.3 (b). The adjacency lists in part (a) of these Figures are now interpreted as cyclically ordered. We denote by $n_f = |F(G, \Pi)|$ the number of faces in an embedding Π of G.

An adjacency list $\Pi_v = u_1, u_2, \ldots, u_d$ may be interpreted also as a list of edges or arcs $(v, u_1), (v, u_2), \ldots, (v, u_d)$. In routine `count_faces` given in Figure 1.4, the input embedding Π is interpreted as a concatenated list of arcs: first list Π_0, then list Π_1, etc. The lists of Π are represented in memory as two n_v by n_v square matrices `prev` and `next` in which row i represents the cyclically ordered list of vertex i (that is, Π_i). The predecessor of vertex x in list i is given by `prev`(i, x), and the successor of vertex

Figure 1.3: $K_{3,3}$ again

Adjacency Lists:

0: 3,4,5
1: 3,4,5
2: 3,4,5
3: 0,2,1
4: 0,1,2
5: 0,1,2

(a)

Faces:

0,3,2,4,0,5,1,3,0,4,1,5,2,3,1,4,2,5,0

(b)

x in list i is given by $\texttt{next}(i, x)$. Thus the successor and predecessor of a given vertex in a given list can be determined (or modified) in constant time, and the existence of a given vertex in a given list can be determined in constant time. The data structure therefore assumes the use of $O(n_v^2)$ space and $O(n_v^2)$ setup time for the lists.

Granting the data structures and setup referred to for Π, the time complexity of `count_faces` is $O(n_e)$ since the concatenated list of Π is traversed twice. But for graphs of fixed genus, or in contexts, such as this thesis, where only graphs of limited edge ratio (see proposition 2.2.2) need be processed, this time complexity translates to $O(n_v)$.

The *genus* of an embedding Π, denoted $g(\Pi)$, is given by the following definition.

Definition 1.1.1 (Generalized Euler's Formula)

$$g(\Pi) = (n_e - n_v - n_f + 2)/2. \tag{1.1}$$

Figure 1.4: The face counting algorithm

```
1. function count_faces(embedding : Π)
       begin
2.     face_count ← arcs_used ← 0
3.     while 2n_e − arcs_used > 0 do
4.         advance face_starter_arc to next unmarked arc in Π
5.         compute Face(Π, face_starter_arc), marking each arc used
               and incrementing arcs_used for each arc in the face
6.         increment face_count
7.     return face_count
       end of count_faces
```

We will refer throughout to this formula as "Euler's Formula". In the topological context this equation requires a proof (such as that by Gibbons [Gib85, p. 72]) since the genus of a surface is defined independently there. But in the combinatorial context, surfaces are not defined, and the formula is used to define the genus of an embedding. On the other hand, in our combinatorial context, it is not obvious that the genus of an embedding is never negative, an issue that will be raised early in the second chapter. The *genus* of a graph G, denoted $g(G)$ is defined to be the minimum of the genera of the embeddings of G. Thus, for example the genus of the embedding given in Figure 1.3 is two, but the genus of the graph there represented is one because the genus of the embedding given in Figure 1.1 is one, and there is no embedding of that graph with lesser genus (with more than three faces). A *planar* graph is one of genus zero, and a *toroidal* graph is one of genus less than or equal one. These definitions accord with the sense of (Generalized) Euler's formula when interpreted in the topological sense. That is, a graph of genus k is one that will embed topologically in the surface with k handles.

The correspondence between embeddings as ordered adjacency lists and as homeomorphisms into oriented surfaces is ascribed jointly to Heffter [Hef91], 1891, and to Edmonds [Edm60], 1960. See, for example Gross and Tucker [GT87]. Youngs [You63] proves that a minimum genus embedding in the topological sense is always an embedding of the type identifiable with a combinatorial embedding.

1.2 History of the Graph Genus Problems

The naturally first question about a graph in this context is " What is its genus?", and the answer follows directly from the very simple face counting algorithm of the preceding section. This, the "naive" algorithm (see Figure 1.5), known to Heffter and

to Edmonds, proceeds by counting the number of faces in the embedding for every possible embedding (every set of circular permutations of the adjacency lists). Euler's formula then gives the genus for that embedding; the genus of the graph is then the minimum of the genera of the embeddings. This algorithm is absolutely impractical except for the very smallest graphs since the number of circular permutations P_v of the single adjacency list for vertex v is $d_v!/d_v = (d_v - 1)!$ where d_v is the degree of v in G. The number of embeddings is therefore $\prod_{v \in V(G)} P_v = \prod_{v \in V(G)} (d_v - 1)!$. Thus the (time) complexity of this, the so-called *naive* algorithm is worse than exponential in the number of vertices.

Thomassen shows in 1989 [Tho89] that the Graph Genus Problem is NP-Complete. Nevertheless, Filotti, Miller and Reif [FMR79] had proved in 1979 that for fixed parameter g, determining whether the genus of a graph is less than g is of polynomial (in the number of vertices) complexity ($O(n_v^{O(g)})$). They give an algorithm for each g. This family of algorithms is the conceptual descendant of an algorithm of Filotti, published in 1980 [Fil80], for embedding *cubic* graphs in the torus, and which is of complexity approximately $O(n_v^{10})$.

Considering the case in which the problem is restricted to genus zero (planar) graphs, a practical algorithm was given in 1964 by Demoucron, Malgrange, and Pertuiset [DMP64], whose time complexity is cubic in the number of vertices. But a variety of practical linear time planar embedding algorithms have since been published and implemented, starting with the one by Hopcroft and Tarjan in 1974 [HT74]. These linear time planarity testing algorithms besides being theoretically very efficient are fast in practice and not only answer the question of the graph genus problem, but also give an embedding when the answer is "yes".

Considering the case in which the problem is restricted to genus one (ie. toroidal) graphs, Filotti's 1980 algorithm gives a genus one embedding when it exists, but its

Figure 1.5: The naive algorithm to determine genus

```
1. function naive_genus(graph : G)
   /*
      An embedding Π of G consists of an adjacency list Π_v for each
      vertex v of G. These lists are interpreted as circular permutations.
   */
      begin
2.    g ←INFINITY
3.    for every circular permutation Π_0
4.        for every circular permutation Π_1
5.            for every circular permutation Π_2
                    ...
n+2.                for every circular permutation Π_{n_v-1}
n+3.                    n_f ←count_faces(Π)
n+4.                    g_Π ← (n_e − n_v − n_f + 2)/2
n+5.                    if g_Π < g then g ← g_Π
n+6. return g
      end of naive_genus
```

limitation to cubic graphs is, until now, real. That is, there is no obvious reduction of the general problem (in which graphs may have vertices of arbitrary degree) to the special case handled by Filotti's algorithm.

Robertson-Seymour theory [RS84, RS85] shows that the time complexity of embedding a graph in the torus is $O(n_v^3)$. Moreover the theory not only states the existence of an algorithm, but can produce one. But the constant of proportionality here is prohibitively large. Nevertheless the existence of such a low degree polynomial theoretically efficient algorithm has fed the hope of finding a practical low degree algorithm. Djidjev and Reif in 1991 [DR91] present another efficient (polynomial time complexity) algorithm for embedding in oriented surfaces, a theoretical improvement over earlier algorithms, but which is not claimed to be useful in practice. As of summer 1992 private communications with experts in the field (N. Robertson, P. Seymour, G. Williamson, C. Thomassen, M. Fellows, D. Archdeacon and others) have not brought to light the existence of a practical, useful algorithm to test toroidality of graphs much less, practically solve the problem for oriented surfaces of higher genus.

1.3 Motivation

Knowing the genus of a graph is interesting for reasons practical as well as theoretical, since genus is a very intuitive way of parameterizing the class of all graphs. Parameterization is of interest because the class of all graphs is too big and amorphous to allow effective algorithmic treatment in many respects. Some graph theoretic problems are known to be efficiently decidable if we restrict to planar graphs. For example, Liu and Chow show [LC83] that counting the number of unicyclic spanning subgraphs of a graph can be done efficiently if the graph is planar. (The number of spanning unicyclic subgraphs of a graph is the second coefficient in the Reliability polynomial,

and the knowledge of it sharpens the estimate of network reliability [Col87]. Interest in determining this number for toroidal graphs was this author's working introduction to the torus and toroidality. An efficient way to solve this problem is still an open question.) The complexity of this problem for the general case is not known.

Some intractable problems when restricted to genus zero, or one, or finite genus become tractable. For example CLIQUE ("Does graph G contain a clique of size k or more?") is solvable in polynomial time when restricted to planar graphs. GRAPH ISOMORPHISM ("Are graphs $G1$ and $G2$ isomorphic?") is an important problem which has been shown to be tractable for finite genus (by Gary Miller [Mil80]). GRAPH ISOMORPHISM in the general case is not currently known to be either solvable in polynomial time or NP-complete. Perhaps algorithms in which genus is fixed would become more common, or at least more investigated when practical embedding algorithms become available. Exploiting such a mode of parameterization we replace the tractable - intractable categorization of problems by an infinite spectrum of tractability, though not for all problems. For example some graph theoretic problems (ref. Garey and Johnson [GJ79]) such as HAMILTONIAN CIRCUIT ("Does graph G have a circuit incident just once to each vertex of G?") and HAMILTONIAN PATH ("Does graph G have a path incident just once to each vertex of G?") are NP-complete even when we restrict to planar graphs.) Fellows and Langston discuss parametrization and applications (in particular to problems commonly arising in VLSI) in a series of papers, for example [FL88].

The torus is increasingly important in physics (knot theory, quantum gravity, fusion reactors) and biology (toroidal units of functionality in the brain), and will inevitably become important in chemistry: "buckyballs", a recently constructed class of spherical crystalline carbon compounds have turned out to be of surprising significance for superconductivity. Toroidal molecules too will eventually be constructed,

and their properties as yet are completely unknown. A practical torus embedding algorithm would seem to be a basic tool in the kit for investigations of the scientific significance of the torus. References to scientific publications in which the torus is important have increased, in the Science Citation Index, to about fifty per year.

Our immediate motivation was to produce a tool that would be useful in a systematic search for the (topological and minor order) obstructions to toroidality. (See Chapter 6 Section 2 for definitions.) Our testing project (Chapter 6) demonstrates a simple minded brute force approach to this application. For example, the 51,662,585 connected, isomorphically distinct graphs on 11 vertices and 23 edges were processed (checked over for topological and minor order obstructions) in a few hundred hours on a single Sun4 processor.

1.4 Algorithm $\mathcal{B}$, a topological sketch

The algorithm sketched in this section has not been implemented nor have the details of implementation been developed. It is the embryo from which the other, implemented, algorithms described in the thesis, have been developed. The idea behind the algorithm is presented in topological terms rather than combinatorial in order to make intuitively clear the arena in which the ideas and algorithms of the thesis proper are combinatorially and rigorously developed.

1.4.1 Graph theory definitions

First we give a few necessary graph theoretic definitions. A *chord* of a subgraph H of G is an edge of G both of whose end vertices are in H but which itself is not in $E(H)$. We denote by $Chords(H)$ the set of chords of a subgraph H with respect to

a containing graph understood within context. If $A \subseteq V(G) \cup E(G)$, then we define $G - A$ to be the graph obtained from G by deleting all vertices of A and all edges in A, as well as all edges incident with vertices of A. If C is an oriented cycle, then $-C$ denotes the same cycle with the opposite orientation. We denote by E_{at} the set of *edges of attachment* of a subgraph H of a graph G understood in context. That is: $E_{at}(H)$ is the set of edges of G with one end vertex in H and the other end vertex in $G - H$. Thus

$$E(G) = E(G - H) \cup E(H) \cup E_{at}(H) \cup Chords(H).$$

The final definition is set theoretic: a *bipartition* of set E is an ordered pair of disjoint sets (E_1, E_2) whose union is E. We say the bipartition is nontrivial if $E_1, E_2 \neq \emptyset$.

1.4.2 The idea of algorithm $\mathcal{B}$

In W. Myrvold's Graph Theory course in the Fall of 1990, student Jim Wall observed that the problem of embedding a graph in the torus is related to the problem of embedding a graph in the plane [1] as follows: If graph G is embeddable in the torus but not in the plane, then there must exist some cycle C of G that, in some toroidal embedding of G, is noncontractible in the torus (C cannot be contracted continuously in the surface of the torus to a point). Otherwise G would be planar. Now if C is such a cycle, then the graph G' remaining when C is deleted from G embeds in the cylinder obtained by cutting along C in the toroidal embedding of G. But the cylinder can be trivially transformed to a sphere by patching discs on the two 'holes' left in the surface cut by C. Thus G' is cylindrical, and so spherical, ie. planar. Moreover, the planar embedding is such that two copies of C can be embedded (oppositely oriented)

[1] This very natural idea has occurred to others—for example Reif and Djidjev, [DR91].

in two faces of G' (the two faces corresponding to the two patches), and such that every edge of G incident to C can be deposited in the planar embedding incident to one or the other copy of C (possibly both, if C has chords).

In fact, as we will show in the following Chapter, it is possible to find a small family of chordless cycles of G such that at least one cycle in the family is noncontractible in some toroidal embedding of G if G is toroidal. Thus we have the foundation of an algorithm to test graphs for toroidality. In order to complete the sketch of algorithm $\mathcal{B}$, Figure 1.6, the following topological (as opposed to combinatorial, ref. Chapter 2) notation is used. If Π is a planar embedding of $G - C$ which has faces F_1 and F_2, then we denote by $\Pi'(C, F_1, F_2)$ the planar embedding obtained from Π by inserting "copies" C_1 and C_2 of (oriented) C inside faces F_1 and F_2 (avoiding putting one copy inside the other if $F_1 = F_2$) giving the two copies opposite orientations. We say that an edge (a, b) of $E_{at}(C)$ with $b \in V(C)$ can be *deposited* in an embedding (containing copies C_1 and C_2 of C) incident to C_i if (a, b') can be deposited in the embedding, where b' is the vertex of C_i corresponding to b. We remark about this algorithm, as here presented, that it lacks the following detail:

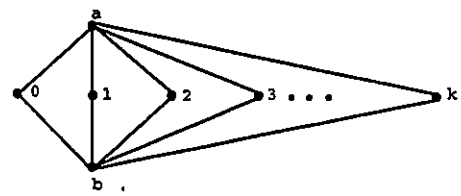
- No routine is specified to generate all planar embeddings of a graph, as is required in step four.
- No data structures are specified for establishing and maintaining orientations.
- No bipartition routine is specified (though algorithms are easily accessible, for example, that in Stanton and White's book [SW86, p. 9]). (Note that generating all bipartitions of a given set is equivalent to generating all subsets of that set.)
- No manner of testing whether the sets E_1 and E_2 are simultaneously embeddable in the given embedding $\Pi'(C, F_1, F_2)$ is specified.

Figure 1.6: Algorithm $\mathcal{B}$ to test toroidality

```
1. function b_toroidal(graph : G)
   begin
2.   generate family F of chordless cycles of G including one noncontractible
3.   for every cycle C of F
4.     for every planar embedding Π of G − C
5.       for every pair (F1, F2) of faces of Π
6.         create Π'(C, F1, F2)
7.         for each nontrivial bipartition (E1, E2) of Eat(C)
8.           if the edges of E1 can be deposited in Π' incident to C1
             and the edges of E2 can be deposited in Π' incident to C2
               then return "TOROIDAL"
9.         create Π'(−C, F1, F2)
10.        for each nontrivial bipartition (E1, E2) of Eat(C)
11.          if the edges of E1 can be deposited in Π' incident to C1
             and the edges of E2 can be deposited in Π' incident to C2
               then return "TOROIDAL"
12.  return "NONTOROIDAL"
   end of b_toroidal
```

Nevertheless, we can say that the complexity of algorithm $\mathcal{B}$, however the foregoing details are implemented, without further refinement, is exponential in n_v. This is because the number of planar embeddings of a given graph is in general exponential in the number of vertices of that graph. (Consider the graph of Figure 1.7 in which the sequence of vertices $(0, 1, \ldots, k)$) embedded in a line, can be embedded in any order — but there are (worse than) exponentially many such orders.)

Figure 1.7: A graph with exponentially many planar embeddings

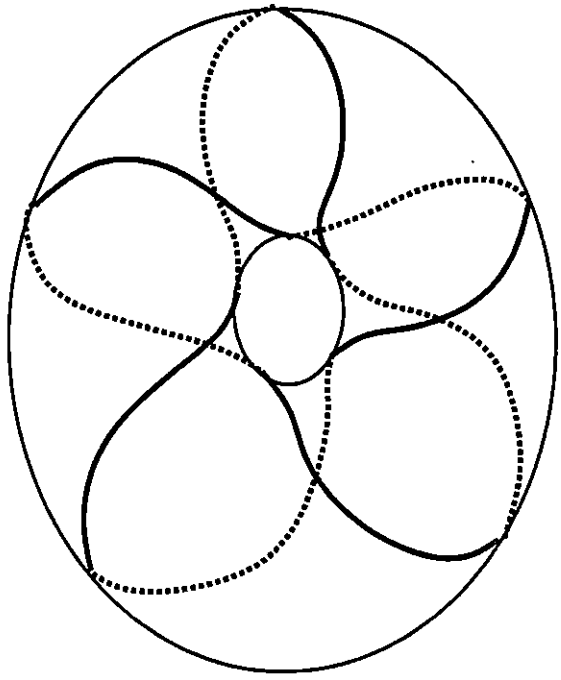


There is another respect in which algorithm $\mathcal{B}$ is exponential, namely, the number of bipartitions of a set is exponential in the size of the set. The size of set $E_{at}(C)$ is linear in n_v. Thus algorithm $\mathcal{B}$ is exponential in n_v. This is obviously not a claimed lower bound on the complexity of the problem, which we already know to be in P, but only a comment on $\mathcal{B}$ as stated.

As a final remark about the idea behind this algorithm (exploitation of a non-contractible cycle) we note that it is a special case of the idea of Reif and Djidjev in [DR91], where two non homotopic noncontractible cycles are first found. (The two cycles not only are not smoothly contractible to a point in the surface, but also, one is not smoothly deformable into the other in the surface.)

Finally, we confirm that the foregoing argument in "intuitive topological terms" with the embryological algorithm does not rigorously validate Wall's idea (which will be done in the following chapter). To underline the absence of rigour here, consider the noncontractible cycle shown in the torus in Figure 1.8. It is perhaps not even

Figure 1.8: An Interesting Noncontractible Cycle



The simple closed curve shown in bold on the surface of the torus is represented in broken line segments as it passes behind.

obvious that the surface remaining after removing the cycle is planar.

1.5 Overview of the Thesis

In the following chapter we give a combinatorial validation of Wall's idea. Algorithm $\mathcal{B}$ clearly does not do it justice. We investigate two divergent developments of the idea, one of which results in an algorithm which improves on the naive algorithm, and is *complete* (gives a definite "yes" or "no" response for every graph input). We label with C this first algorithm, presented in Chapter 3.

The other development is a "proof by contradiction" approach to demonstrating

nontoroidality: a graph is assumed to be toroidal and a contradiction is derived, or, in some cases, a toroidal embedding is produced. This latter approach results in a much faster algorithm which, however, sometimes returns the answer "Inconclusive", and so is incomplete. This algorithm (labelled $\mathcal{D}$) is presented in Chapter 4.

There are various approaches to combining $\mathcal{C}$ and $\mathcal{D}$ The algorithm obtained by combining $\mathcal{D}$ and $\mathcal{C}$ in tandem, so that $\mathcal{D}$ is a preprocessor for $\mathcal{C}$, though complete (never returns "Inconclusive") and much faster than $\mathcal{C}$ on average, can still be greatly improved by a more profound unification of the two approaches. The unified algorithm, still exponential but having some practical value, which we may call $\mathcal{E}$ is presented in Chapter 5.

We demonstrate the speed and effectiveness of the final implementation $\mathcal{E}$ by computing the two lists of topological and minor order obstuctions to toroidality on up to 10 vertices, and testing several long lists of larger graphs (11 to 14 vertices) of size $\approx 10^8$ graphs. We show incidentally that minor obstructions (see Chapter 6 Section 2 for the definition) on graphs with 12 and 13 vertices are easy to find: Twenty hours of Sun4 time revealed 37 twelve vertex minor order obstructions to toroidality. The same amount of time revealed a single 13 vertex obstruction to toroidality. (Eight minor order obstructions on 13 vertices were found in all. Since the number of graphs examined were a random segment of size a very small fraction of one percent of the number of all 13 vertex graphs, there could be more.) Several hundred hours revealed no 14 vertex obstructions, but this should not be taken as a basis for conjecture, for example that 13 is an upper limit on minor order obstruction size.

Algorithm $\mathcal{E}$ does not readily generalize to higher genus testing. The methods we present for certifying nontoroidality as algorithm $\mathcal{D}$ are generalizable to higher genus.

Chapter 6 summarizes the computational experience and includes tables of obstructions to toroidality. In Chapter 7 we consider future extensions.

Chapter 2

Validation of the Germinal Idea

In this chapter we validate combinatorially the idea that we can test toroidality by finding a family of chordless cycles guaranteed to include a noncontractible cycle, and solving some problems about the planarity of certain auxilliary graphs. In a preliminary first section we give some definitions and prove some basic facts about embeddings. In the second section we show that solving a restricted version of the problem does not lose generality. In the third section we validate Wall's idea by defining contractibility and deriving some results about contractible cycles. In the final section we consider families of cycles with a view to guaranteeing inclusion of a chordless noncontractible cycle.

2.1 Basic Facts about Embeddings

In the introduction we used the generalized Euler's formula, Equation 1.1 to define genus. However, nothing is for free. In this, the combinatorial context, since surfaces are not preexisting objects, it is necessary to show that graphs cannot have negative

genus. We follow Thomassen [Tho90, page 158] for this development. A graph is *connected* if it includes a path between every two vertices, otherwise it is *disconnected.* If Π is an embedding of connected graph G which has connected subgraph H, then the *induced embedding* of H (also referred to as Π) is obtained from that of G by ignoring all the edges not in H.

Proposition 2.1.1 (Thomassen, [Tho90, page 158]) *If e is an edge of connected G such that $G - e$ is connected, then the genus of $G - e$ is equal to or one less than the genus of G.* □

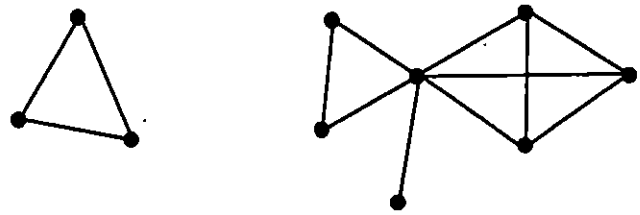
Proposition 2.1.2 (Thomassen, [Tho90, page 158]) *The genus of any connected graph is zero or positive.* □

2.2 Restricting the Problem

In this section we demonstrate several ways of reducing the problem to special cases without losing generality, by virtue of some well known theorems. A subset A of edges and vertices of connected graph G is *separating* if $G - A$ is disconnected. G is a *block* or *two-connected* if there does not exist vertex v in G such that $G - \{v\}$ is disconnected. A *component* of a graph is a maximal connected subgraph. We use n_c to denote the number of components of a graph. A *block* of a graph G is a maximal block in G. We use n_b to denote the number of blocks of a graph. The graph of Figure 2.1 has two components and four blocks.

Because of the following result presented here without proof, we can in this thesis restrict our attention to determining toroidality of blocks.

Figure 2.1: A disconnected graph with four blocks



Theorem 2.2.1 (Battle,Harary,Kodama,Youngs [BHKY62]) *The genus of any graph is the sum of the genera of its blocks.* □

Restricting attention to blocks means, in particular, that the graphs we test will have no vertices of degree zero or one. We can also restrict our attention to graphs with no vertices of degree two, since it is obvious that any vertex of degree two in an embedding can be replaced, together with its two incident edges, by a single edge, leaving the genus unchanged. (n_f is unchanged, n_v decrements, and n_e decrements, so by Euler's Formula, Equation 1.1 $g(\Pi)$ is unchanged.) Graphs related by such replacements are called *homeomorphs* or *homeomorphic*. Reducing a degree two vertex can trigger further reductions, since multiple edges may appear and be deleted as they appear. A graph which has been reduced to the point where it has no degree two vertices and no multiple edges is called *series parallel reduced.*

A very useful fact about toroidal graphs is expressed in the following well known proposition (compare eg. Gibbons [Gib85, Cor 3.1,p.70] which however is less general) from which it follows that we may restrict attention, in testing for toroidality, to graphs for which $n_e \leq 3n_v$. If this inequation is not satisfied for a particular graph we say that the *edge ratio* (of three) is *exceeded*, and imply thereby that G cannot be toroidal.

Proposition 2.2.2 (compare Gibbons [Gib85, Cor. 3.1, page 70]) *If connected graph G has genus g then $n_e \leq 3n_v - 6 + 6g$.*

Proof: Let Π be a genus g embedding of G. Since every edge of G is included just twice in the faces of Π, we know that the sum over all faces F_i of Π of the number of edges $|F_i|$ of F_i must equal twice the number of edges: $\sum_{F_i \in F(\Pi)} |F_i| = 2n_e$. Since every face has at least three edges we also have that $\sum_{F_i \in F(\Pi)} |F_i| \geq 3n_f$. Therefore $n_f \leq 2n_e/3$. Substituting in Euler's Formula, Equation 1.1 produces $n_e \leq 3n_v - 6 + 6g$. □

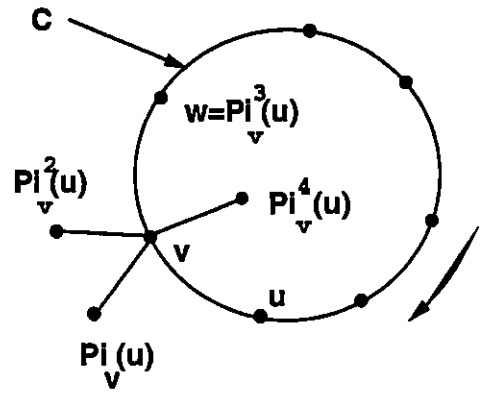
Another consequence of using this result to restrict the domain of graphs tested is that operations of time complexity linear in n_e become linear in n_v whether the graph studied is toroidal or not. It is noteworthy that graphs of *any* fixed genus have $n_e = O(n_v)$ even though graphs in general have $n_e = O(n_v^2)$. This is a simplest example of the amelioration of complexity resulting from parameterization of graphs by genus.

The routine to carry out series-parallel reduction on a graph is called `ser_par_red`. If a degree vector is maintained, and the vertices are processed in order from 0 to $n-1$, and a stack of "current vertices" is maintained, with the current vertex being pushed onto the stack whenever it is necessary to reprocess an earlier vertex, then the time complexity of this routine is $O(n_v)$ since each reduction can be done in constant time, and at most n_v reductions are performed while proceeding in order through the vertices of G and "stepping back" only when an earlier vertex must be reduced, and then returning to the current position. (The number of times the stack is augmented is $O(n_v)$ since reprocessing an earlier vertex occurs only when a reduction is performed.)

2.3 Contractibility

In this section we present some definitions and results about contractibility, a crucial concept in our algorithms. In order to define *contractible* we need some definitions and a fundamental proposition of Thomassen proved in [Tho90]. If C is a cycle, considered oriented, of G embedded by Π, (see Figure 2.2) and if (u, v) is an edge of C followed

Figure 2.2: An Embedded Graph Fragment



by (v, w), then $w = \Pi_v^p(u)$ for some positive integer p. All edges $\Pi_v(u)$, $\Pi_v^2(u)$, ..., $\Pi_v^{p-1}(u)$ are said to be on the *left* side of C, while the edges $\Pi_v^{p+1}(u)$, $\Pi_v^{p+2}(u)$, ..., $\Pi_v^{d-1}(u)$, (where d is the degree of v) are said to be on the *right* side of C. In the graph fragment of Figure 2.3, if the cycle is oriented $(0, 4, 5, ...)$ and $\Pi_4 = (0, 3, 1, 5, 6, 8, 9)$ then $(3, 4)$ and $(1, 4)$ are on the left side of the cycle, while $(6, 4)$, $(8, 4)$, and $(9, 4)$ are on the right side. Any edge in G not incident to C, connected by path to a left edge of C (such as $(2, 1)$) is also said to be on the *left* side of C. Thus, in the graph fragment of Figure 2.3, all the edges incident to vertices 1 and 3 are on the left side of C. The *left graph* of a cycle C of a graph G, embedded by Π, denotable $G_l(C, \Pi, G)$ is then defined to be the edges on the left side of C together with all their end vertices. When context permits, $G_l(C, \Pi, G)$ is abbreviated to $G_l(C)$. Graph $G_l(C) \cup C$ is the

subgraph of G consisting of all the vertices and edges of $G_l(C)$ and of C. Similarly we can define all the above for the right side of C. The following proposition given by Thomassen demonstrates that contractible cycles behave according to our topological intuition, and is referenced directly or indirectly in the justification of algorithms $\mathcal{C}, \mathcal{D}$ and $\mathcal{E}$.

Proposition 2.3.1 (Thomassen [Tho90, Proposition 3.2, page 159]) *If C is a cycle in a Π-embedded graph G with disjoint (except for vertices on C) left and right graphs $G_l(C, \Pi, G)$ and $G_r(C, \Pi, G)$ then the Π-genus of G is the sum of the Π-genera of $G_r(C) \cup C$ and $G_l(C) \cup C$.* □

In the terminology of Proposition 2.3.1, if one of $G_r(C) \cup C$ and $G_l(C) \cup C$ has Π-genus zero, we say that C is Π-*contractible*, and we call the side (G_r or G_l) with Π-genus zero the Π-*interior* or just *int* of C. The other side is called the Π-*exterior* of C or just *ext* of C. Both the left graph unioned with C and right graph unioned with C could have genus zero. In this case we fix one side to be the interior and the other to be the exterior. If neither has genus zero, then there is no interior nor exterior of C in G with respect to Π and we say C is Π-noncontractible. In particular, if G is a nonplanar graph, and $G_l(C) = G_r(C)$, then C is noncontractible.

The terms *Int* and *Ext* have similar denotations to (resp.) int and ext, except that in each case, the cycle is included in the subgraph.

If for some cycle C of G, and some toroidal embedding Π of G we have $G_l(C) = G_r(C)$ then we label $G_{\Pi CC}$ the graph obtained from G by "duplicating and splitting" (formal definition follows) C in Π. Figure 2.4 illustrates (a fragment of) the duplicated and split cycle of the embedded fragment of Figure 2.3. We obtain graph $G_{\Pi CC}$ from graph G by *splitting and duplicating* cycle C of G when $G_l(C, \Pi, G) = G_r(C, \Pi, G)$ as follows: Create a cycle C' isomorphic to C with vertices x' corresponding to x. For

Figure 2.3: Another Embedded Graph Fragment

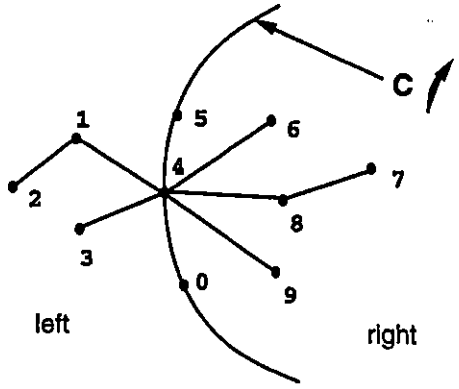
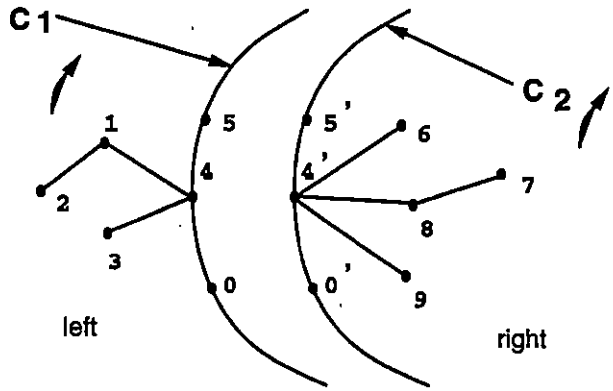


Figure 2.4: Duplication and splitting of C



vertex x in C, remove all edges (x, v) in $G_r(C)$, inserting new corresponding edges (x', v). In the resulting graph we may call C C_1, and C' C_2. The graph $G_{\Pi CC}$ inherits an induced embedding from Π with essentially the same faces as Π plus two new faces C_1 and C_2.

Proposition 2.3.2 *If graph G is toroidal, and cycle C in G is noncontractible in some embedding Π of G in the torus, then $G_{\Pi CC}$ is planar.*

Proof: If $G_r(C, \Pi, G)$ and $G_l(C, \Pi, G)$ are edge disjoint, both $G_l(C) \cup C$ and $G_r(C) \cup C$ must have genus at least one since C is noncontractible. But then G has Π-genus equal to their sum, greater than one, a contradiction. Therefore $G_r(C, \Pi, G) = G_l(C, \Pi, G)$. Graph $G_{\Pi CC}$ with the induced embedding has all the vertices, edges and faces of G (embedded by Π) plus the $n_c = |V(C)|$ new vertices and $|E(C)| = |V(C)|$ new edges of the "other" copy of C, plus the two new faces. Applying Euler's formula gives

$$g(\Pi, G_{\Pi CC}) = (n_e + n_c - n_v - n_c - n_f(\Pi, G) - 2 + 2)/2$$

but since G is toroidal we have also

$$1 = (n_e - n_v - n_f(\Pi, G) + 2)/2$$

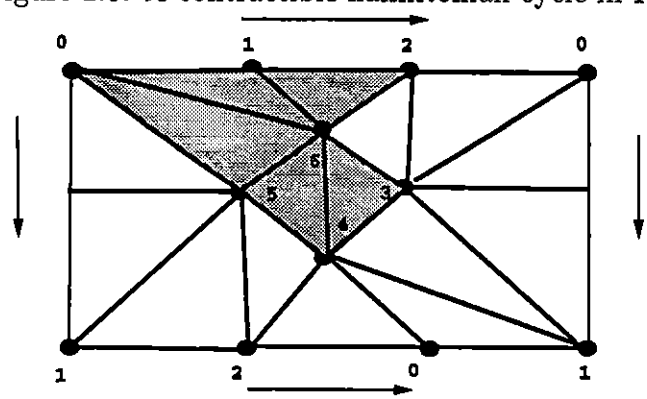
and therefore $g(\Pi, G_{\Pi CC}) = 0$ and so also $g(G_{\Pi CC}) = 0$. □

The foregoing propositions complete the justification of Wall's essential idea, as well as suggesting that a test for toroidality need not involve generating every planar embedding of $G - C$: the existence of a planar graph abstractly identical to $G_{\Pi CC}$ with the embedding properties of $G_{\Pi CC}$ as embedded by Π assures the toroidality of G, indeed produces a toroidal embedding of G (by reidentification of the two cycles). And the nonexistence of such an auxilliary graph having such an embedding assures the nontoroidality of G. Note however that the mere existence of a planar embedding

of abstract $G_{\Pi CC}$ does not prove the toroidality of G. (A counter example could be constructed by taking a large dense grid (nontoroidal) on the Klein bottle and splitting and duplicating a noncontractible cycle.) So the problem becomes a matter of finding a noncontractible cycle, and then somehow determining if an embedding *like* that described above exists. This problem is addressed in Chapter 3.

We note in passing that there exist toroidal graphs with a cycle C contractible in some toroidal embeddings, and noncontractible in other toroidal embeddings. It follows from Proposition 2.3.2 that if C is a noncontractible cycle, then $G-C$ is planar. But even when G is toroidal nonplanar and $G - C$ planar it does not follow that C is noncontractible in every toroidal embedding. Consider for example a Hamiltonian cycle C in K_7 (a toroidal graph with seven vertices and twenty-one edges): $K_7 - C$ is empty and so planar, but it is easy to find an embedding of K_7 in which the cycle is contractible. See Figure 2.5. Worse: even when cycle C of toroidal nonplanar G is such that $G - C$ is planar in *every* toroidal embedding of G, it does not follow that C is noncontractible in some toroidal embedding of G (see Section 4.2.2, Figure 4.3). Therefore when we seek a noncontractible cycle, it is not enough to find a cycle C such that $G - C$ is planar.

Figure 2.5: A contractible hamiltonian cycle in K_7



2.4 Fundamental Sets of Cycles

Any torus embedding algorithm exploiting Wall's idea must produce a cycle or a small family of cycles with the property that at least one of them is noncontractible in some embedding of the graph in the torus, if G is toroidal. But we know of no way to find a noncontractible cycle of a toroidal graph short of first finding a toroidal embedding of the graph. Therefore in this section we consider ways of generating small families of cycles guaranteed to include a noncontractible cycle if one exists.

Bases and fundamental sets of cycles are important in both algorithms $\mathcal{D}$, and $\mathcal{C}$. In this subsection we define them, and prove the proposition that a fundamental set of cycles (indeed any basis of the cycle space) of a toroidal graph must include a noncontractible cycle of any toroidal embedding of the graph.

2.4.1 Definitions

A *tree* is a connected graph that has no cycles. A *leaf* in a tree, or any graph, is a vertex of degree one. A *root* of a tree is a unique arbitrarily selected vertex of the tree. A *spanning tree* T of a graph G is a subgraph of G which is a tree, and includes every vertex of G. A spanning tree has $n_v - 1$ edges. We orient the edges of a tree so that all paths are directed toward the root. The edges of G which are not in a spanning tree T of G, are each such that inserting the edge into the tree produces a unique cycle of G [Gib85, p.7].

If T is a spanning tree of G, the set of $n_e - n_v + 1$ edges $E(G) - E(T)$ determine a set F of $n_e - n_v + 1$ distinct cycles called a *fundamental set* of cycles of G. It is a fact that a set of fundamental cycles constitutes a basis for the cycle space (considered as a vector space over the two element field) of a graph [Gib85, p.55].

2.4.2 Breadth First Search

Routine `breadth_first_search` is a routine which given a graph G with selected "root" vertex r returns, in time linear in the number of edges, a tree for G rooted at r with the property that the path from r to v in the tree is a shortest path (in G), for every vertex v of G. Such a *breadth first search tree* or *BFS*-tree of a graph determines a fundamental set of cycles some of which may have chords. The fundamental set of cycles can be stored at no extra (theoretical) cost simply as a list of non-tree-edges. The unique parent in the BFS tree, of vertex v is represented by $BFST(v)$. Each cycle is stored as a non-tree-edge of G and can be listed by following the branches starting at v and at w back to their first common ancestor, at worst the root of the tree. Since all non-tree-edges either have both end vertices at the same depth in the tree or at depths differing by one (this is the nature of a breadth first search tree), the first common ancestor can be determined by comparing $BFST(w)$ and $BFST(v)$, and stepping back through the parents of w and v, and comparing parents at each step.

2.4.3 Guaranteeing Inclusion of a Noncontractible Cycle

The following proposition asserts that any fundamental set, and in general any basis of the cycle space has the required property. The family, as we have shown, is of size $O(n_e)$, and therefore, for the graphs we test, of size $O(n_v)$.

Proposition 2.4.1 *If $\mathcal{F}$ is a basis of the cycle space of nonplanar block G, and if Π is a toroidal embedding of G, then some $C \in \mathcal{F}$ is noncontractible in Π.*

Proof: Let S be the smallest subspace of the cycle space of G including all cycles contractible in Π. Then S is contained in the space generated by the faces of Π.

(Indeed, let C be a cycle of G contractible for Π. Then in the planar embedding of $Int(C)$ induced by Π, since the sum of all faces is null, C is the sum of the other faces, all of them faces also of Π.) But S has dimension at most $n_f = |F(G,\Pi)| = n_e - n_v$, (Euler's Formula, Equation 1.1 with $g = 1$). Since a fundamental set is a basis of the cycle space [Gib85, p.55] every basis of the cycle space must have $n_e - n_v + 1$ independent elements. But then some element of the basis cannot be in S. That is, some cycle of the basis must be noncontractible in Π. □

2.4.4 Smaller Families of Cycles

The results given in this subsection are not used in our algorithms but are relevant for two reasons. First they show how a very small family of candidate cycles guaranteed to include a noncontractible cycle can be found in linear time Second, Kuratowski graphs will be introduced, which are are an example of the obstructions, at a lower level (they are obstructions to planarity), for which we search, as a test of our algorithms, described in a later chapter.

In the following famous theorem of Kuratowski. $K_{3,3}$ is the graph of Figure 1.1, and K_5 is the *complete* graph on five vertices—that is, the graph on five vertices containing every possible edge:

Theorem 2.4.2 (Kuratowski's Theorem [Kur30]) *A graph is planar if and only if it contains no homeomorph of $K_{3,3}$ or K_5.*

Proof: See for example Bondy and Murty [BM76, pages 153–156]. □

Equivalently we may say that every nonplanar graph contains a homeomorph of either $K_{3,3}$ or K_5. These two graphs, and their homeomorphs, are called *the Kuratowski*

graphs. The fact about Kuratowski graphs relevant to our problem is stated in the following theorem.

Theorem 2.4.3 *If $\mathcal{F}$ is a cycle basis of a Kuratowski subgraph K of toroidal G, then some cycle C in $\mathcal{F}$ is noncontractible in some toroidal embedding Π of G.*

Proof: Let Π be any toroidal embedding of G. Then the induced embedding of K will be toroidal, because K is toroidal but nonplanar. By proposition 2.4.1 $\mathcal{F}$ includes a cycle C which is noncontractible in the induced embedding of K. Therefore C is noncontractible also in embedding Π of G. □

But a set of fundamental cycles of a Kuratowski subgraph contains very few elements. In the case of (any homeomorph of) $K_{3,3}$, four; and in the case of (any homeomorph of) K_5, six.

Williamson [Wil85] shows how to find a Kuratowski subgraph of a nonplanar graph in $O(n_v)$ time using a refinement of the Hopcroft Tarjan [HT74] planar embedding algorithm. We conclude that it is possible, in linear time, to find a family of at most six cycles, guaranteed to include a noncontractible cycle in any toroidal embedding of a given nonplanar graph.

Chapter 3

Algorithm $\mathcal{C}$, a refinement of $\mathcal{B}$

In this chapter we present toroidality testing algorithm $\mathcal{C}$ after a theoretical discussion. The algorithm employs a modification of the planar embedding algorithm of Demoucron, Malgrange and Pertuiset [DMP64], so the first section is dedicated to a review of that algorithm. In the second section we establish the primary theory underpinning the toroidality testing algorithm. In the third section we reconsider the problem of generating a family of suitable cycles guaranteed to include a noncontractible cycle in any toroidal embedding of a graph. In the final section we give pseudo code for algorithm $\mathcal{C}$. This chapter owes much to Gibbons [Gib85], and the first section in particular, presenting the planar embedding algorithm, follows Gibbons.

3.1 The Planar embedding Algorithm

3.1.1 Restricting Input

Corresponding to the limiting edge ratio of three in the case of toroidality, we know in the case of planarity, by proposition 2.2.2 letting $g = 0$ that $n_e > 3n_v - 6$ implies

the graph is nonplanar. Thus we also can restrict attention to graphs for which $n_e \leq 3n_v - 6$. As in the case of toroidality testing, here too we could restrict attention to input graphs that are blocks with no vertices of degree less than three. Also, by Kuratowski's theorem 2.4.2, graphs with fewer than nine edges or five vertices must be planar, so we could restrict attention to graphs with at least five vertices and at least nine edges. But our main application of the planar embedder, though always processing blocks, may receive input graphs with degree two vertices.

3.1.2 Definitions

If H is a subgraph of G and P is a component of $G - H$ or a chord of H, then we call the subgraph of G consisting of P together with all edges incident to P, and their end vertices a *bridge* B of G (with respect to H) *only if* P is adjacent (incident, when a chord) to at least two vertices of $G - P$. For example the graph G in Figure 3.1 has just two bridges with respect to the cycle $(0, 1, 2, 3, 4)$, drawn separately as $B1$ and $B2$. The *vertices of attachment* of B are $V_{at} = V(B) - V(P)$. The *edges of attachment* of B are $E_{at} = E(B) - E(P)$ unless P is a chord, in which case $E_{at} = P$. In Figure 3.1, V_{at} for $B1$ is $\{0, 1, 4\}$, E_{at} for $B1$ consists of three edges, V_{at} for $B2$ is $\{1, 4\}$, and E_{at} for $B2$ is $\{\{1, 4\}\}$. One of the two components of $G-H$ in the diagram does not determine a bridge because it would have only one vertex of attachment. (Since only blocks are input to the planarity tester, such a component never arises in our application.)

Let Π be a planar embedding of subgraph H of planar G. If Π can be extended to a planar embedding of G then it is called *G-admissible.* That is, Π is a G-admissible embedding of H if it is a planar induced embedding with respect to some planar embedding of G. For example the subgraph consisting of the graph $G - \{(1, 3)\}$ of

Figure 3.2 has the inadmissible embedding (b) and the admissible embedding (c).

We say bridge B of G with respect to H can be *drawn* in face f of Π if for B, $V_{at} \subseteq V(f)$ (regardless of the genus or complexity of B) We denote by $Faces(B, \Pi)$, the faces of Π in which B is drawable. And so in Figure 3.2, if we call the embedding in (b) Π, and the graph consisting of edge $\{1,3\}$ and its end vertices B, we have $Faces(B, \Pi) = \emptyset$.

3.1.3 The algorithm

The planar embedding algorithm `planar`, Figure 3.3, operates on the assumption that the input graph G is planar, and if G is indeed planar, then each of the sequence of planar embeddings Π_i of G_i is G-admissible and the final embedding is output as a planar embedding of G. If the input graph is in fact nonplanar then a contradiction is derived when at some stage we find that for some bridge B with respect to G_i embedded by Π_i, $Faces(B, \Pi_i) = \emptyset$. The routine is terminated at step six when a contradiction to planar embeddability has been found, signalled by $EMBEDDABILITY$ = **false**, or when a planar embedding of G has been built, signalled by the number of faces in the current embedding $f = n_e - n_v + 2$, in accord with Euler's Formula, Equation 1.1 with $g = 0$.

We note that `planar` is low degree polynomial: The while loop is executed $O(n_v)$ times. Since finding connected components can be done (via depth first search, for example) in linear time, bridges too can be found in linear time. For a given bridge, the set of $O(n_v)$ faces in which it can be drawn can be determined in at most $O(n_v^2)$ time. No step from 11 to 16 takes more than linear time. So `planar` is at worst $O(n_v^4)$.

Figure 3.1: Bridges

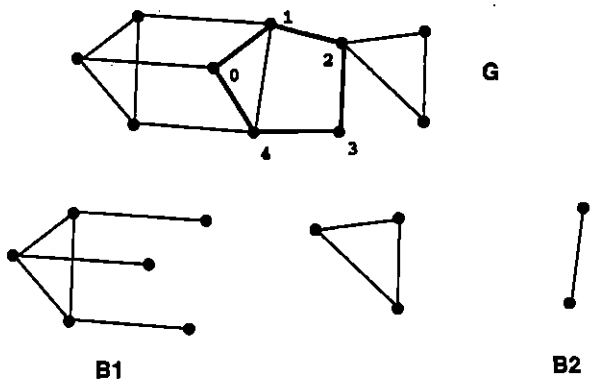


Figure 3.2: Admissibility

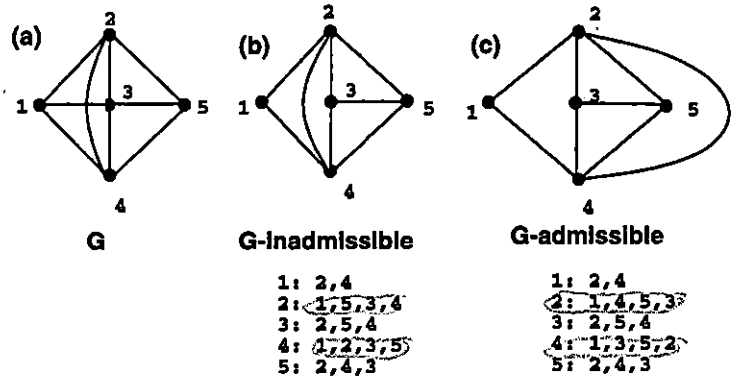


Figure 3.3: The planarity testing algorithm

```
1. function planar(graph : G)
     begin
2.   Find a cycle C of G
3.   i ← 1, G_1 ← C, Π_1 ← C
4.     f ← 2    /* f is current number of faces */
5.     EMBEDDABLE ← true /* C is embeddable in plane */
6.     while f ≠ n_e − n_v + 2 and EMBEDDABLE do
         begin
7.       find each bridge B of G with respect to G_i
8.       for each B find Faces(B, Π_i)
9.       if for some B, Faces(B, Π_i) = ∅ then
           EMBEDDABLE ← false, return false
10.      if EMBEDDABLE then
11.        begin
12.        if for some B, |Faces(B, Π_i)| = 1 then F ← Faces(B, Π_i)
13.        else let F be any face in any bridge F ∈ Faces(B, Π_i)
14.        find a path P_i in B connecting two vertices of V_at for B
15.        insert P_i in face F of Π_i to get Π_{i+1} of G_{i+1}
16.        increment i and f
           end of if EMBEDDABLE
         end of while
17.    return true
     end of planar
```

3.2 Adaptation of the planar algorithm

In proposition 2.3.2 we established that if C is a noncontractible cycle in toroidal G, that $G_{\Pi CC}$ is planar. If C is also chordless, the graph $G_{\Pi CC}$ determines a bipartition (E_1, E_2) of the edges E_{at} of C, where E_i is the set of edges of E_{at} incident to copy C_i (or, in G, left and right of C). Note that neither subset of the bipartition is empty (or G would be planar), but that one of the two classes may have as few as one edge. The abstract graph $G_{\Pi CC}$ is therefore determined by G, C, E_1, and E_2 and thus may be denoted $G(C, E_1, E_2)$.

3.2.1 The special planar embedding sought

We here recall that not every planar embedding of $G(C, E_1, E_2)$ corresponds to an embedding $G_{\Pi CC}$. In order to derive the toroidal embedding Π of G we seek, (by reidentifying corresponding vertices of C_1 and C_2 as described in the second chapter) the planar embedding of $G(C, E_1, E_2)$ we find must be such that C_1 and C_2 are faces, and oppositely oriented. The following proposition expresses the fact we need.

Proposition 3.2.1 *Chordless cycle C of nonplanar block G is a noncontractible cycle in a toroidal embedding of G if and only if there exists a bipartition (E_1, E_2) of the edges E_{at} of C such that graph $G(C, E_1, E_2)$ has a planar embedding in which the two oppositely oriented copies of C are (oriented) faces.*

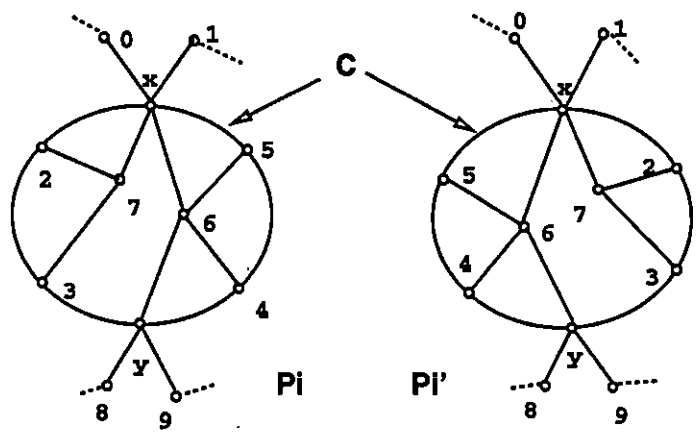
Proof: If no such planar embedding exists, then by proposition 2.3.2, no toroidal Π of G exists in which C is noncontractible. On the other hand, if such a planar embedding does exist, then constructing a toroidal embedding by reidentifying the two cycles produces a toroidal embedding of G using the same application of Euler's equation as was used in Proposition 2.3.2. □

We must now specify how it can be determined whether graph $G(C, E_1, E_2)$ has a planar embedding in which C and $-C$ are (oriented) faces. For this we need a fact about *2-switching* which we discuss next.

3.2.2 2-switching

Whitney [Whi33] proved that any two planar embeddings of a two-connected graph are related by a sequence of 2-switchings. Thomassen gives a combinatorial exposition of the crucial concept in [Tho90, Section 6] which we reproduce here. (See Figure 3.4 in which a 2-switching of C produces Π' from Π.) Let C be a Π-contractible

Figure 3.4: 2-Switching



cycle of block G, and suppose that only two vertices x and y of C are incident with edges in $ext(C)$. Then we define a new embedding Π' as follows: For each vertex z in $ext(C, \Pi) - \{x, y\}$, the Π'-clockwise ordering around z is the same as the Π-clockwise ordering. For each z in $Int(C, \Pi) - \{x, y\}$, the Π'-clockwise ordering around z is the Π-counterclockwise ordering around z. If $e_1, e_2, \ldots, e_k, e_{k+1}, \ldots, e_d$ is the Π-clockwise ordering around x (resp. y) such that $(e_1, \ldots, e_k)$ are in $Int(C, \Pi)$ and $(e_{k+1}, \ldots, e_d)$ are in $ext(C, \Pi)$ then the Π'-clockwise ordering around x (resp.

y) is $(e_k, e_{k-1}, \ldots, e_1, e_{k+1}, \ldots, e_d)$. We say that Π' has been obtained from Π by a 2-switching of C, and that the edges in $Int(C, \Pi)$ are *involved* in that 2-switching. Only two facial cycles are affected by the 2-switching (not counting orientation of faces), and the total number of faces is unchanged, so the genus is unchanged. Intuitively, the disk bounded by the cycle C in the surface of the embedding is pivoted 180 degrees about the axis determined by x and y. See Figure 3.5 for pseudocode for routine `flip`.

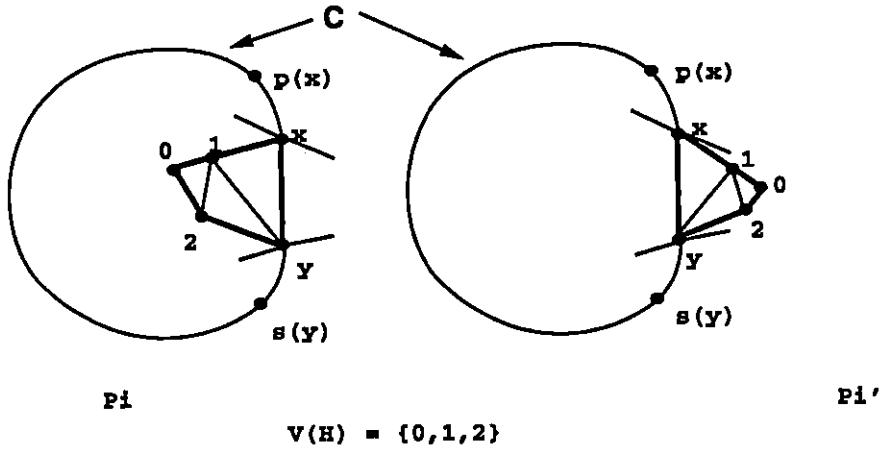
Figure 3.5: Routine to perform 2-switching

1. **function** `flip`(*embedding* : Π; *cycle* : C; *graph* : G; *vertices* : x, y)
   /* x and y are vertices of Π-contractible C */
   /* $\{x, y\}$ is a cut set of G, the embedded graph */
   begin
2. **for** v in $int(C) - \{x, y\}$ invert list of v in Π
3. find the first and last vertices $\{u, v\}$ in Π_x belonging to $Int(C)$
4. invert sublist (u, v) of Π_x
5. find the first and last vertices $\{u, v\}$ in Π_y belonging to $Int(C)$
6. invert sublist (u, v) of Π_y
   end of `flip`

We need the following proposition (illustrated by Figure 3.6 in which we show a 2-switch of cycle $(0, 1, x, y, 2)$ (in boldface).)

Proposition 3.2.2 *If C is a Π-contractible cycle of block G, and component H of $G - C$ is contained in $int(C, \Pi)$, and H is adjacent to only two successive vertices x*

Figure 3.6: A genus preserving transformation



and y of C then there exists a 2-switching leaving C contractible and moving H to $ext(C, \Pi')$.

Proof: Let Π be as in the hypothesis. We can suppose without loss of generality that $int(C)$ is $G_r(C)$ and C is oriented so that y is the successor of x. The cycle to use for the 2-switch is $Face(\Pi, (x, y))$ of the induced embedding of the subgraph induced by $H \cup \{x, y\}$ (boldface in Figure 3.6). □

3.2.3 Ensuring the cycle copies are faces

As a first step to showing how it can be determined whether graph $G(C, E_1, E_2)$ has a planar embedding in which C and $-C$ are (oriented) faces, we define a *wheel* to be a graph consisting of a cycle C, called the *rim*, and a *hub* vertex $v \notin V(C)$ adjacent to every vertex of C by a *spoke* edge, and we prove the following

Proposition 3.2.3 *If wheel (C, v) is a subgraph of block G embedded in the plane by Π and v is adjacent in G only to vertices of C then there is a planar embedding of $G - v$ in which C is a face.*

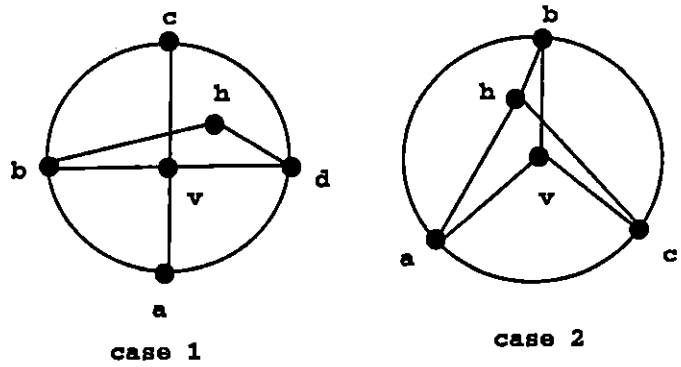
Proof: By hypothesis, C is contractible in Π. We will call $int(C)$ the side containing component $\{v\}$. Then the other components of $G - C$ in $int(C)$ are each adjacent to at least two vertices of C (G is a block), but to at most two vertices of C, and those successive in C.

Topologically, this last assertion is a consequence of the Jordan curve theorem. Combinatorially, we argue first that an interior component H can be shrunk to a single vertex h without altering the genus of the embedding.

Secondly, supposing that H was adjacent to more than two vertices of C, we show that every possible configuration of ordered adjacency lists for the vertices of the subgraph induced by h, the vertices in C adjacent to h, and v, produce an embedding of genus greater than zero, a contradiction. Specifically, (see Figure 3.7), there are two general cases to consider: case 1. h is adjacent to two nonadjacent vertices b and d of C, (and possibly others). Then the induced planar subgraph homeomorph with vertices v, h, a, b, c, d, where a, b, c, d are vertices of C, in order, and h is adjacent to b and d, must include face (unordered) (a, b, c, d). There are four possible embeddings of the graph induced by these six vertices subject to the given constraints. All have genus one: a contradiction. case 2. C has three vertices and h is adjacent to all three, so that the induced planar embedding of the homeomorph with vertices h, v, a, b, and c, where a, b, c are the vertices of C adjacent to both h and v, must include face (unordered) (a, b, c). There are eight possible embeddings of the graph induced by these five vertices, and all have genus one, a contradiction. Thus H is adjacent to just two successive vertices of C.

Therefore by proposition 3.2.2, H can be 2-switched to $ext(C)$. This applies to every component H in $int(C)$ successively. When every such H has been 2-switched to $ext(C)$ giving planar Π', since $\{v\}$ is the only other component of $int(C)$, we see that the induced embedding of $G - v$ has $int(C)$ empty, that is, C is a face in

Figure 3.7: Combinatorial Jordan curve argument



$\Pi'(G - v)$. □

By proposition 3.2.3, we can say that $G' = G(C, E_1, E_2)$ has a planar embedding in which C_1 and C_2 are faces if G'' consisting of G' augmented by two extra vertices, one adjacent to every vertex of C_1 and the other adjacent to every vertex of C_2 is planar. (Although G' is not necessarily a block, each cycle must occur inside a single block, and so the blocks may be dealt with individually). Routine `build_G''` is given in Figure 3.8.

3.2.4 Finding an appropriate planar embedding of G''

We seek a planar embedding of the auxilliary graph G' in which the two copies of C are oppositely oriented faces. In the preceding section we show when a planar embedding of G' exists in which the two copies of C are faces. Unfortunately, C_1 and C_2 may have the same orientation (seen as copies of C). Our second step is to establish a test that enforces opposite orientations. That is, we are only interested in the existence of a planar embedding of G'' in which C_1 and C_2 are oppositely oriented (as faces: that is, in the planar embedding of $G'' - \{v_1, v_2\}$ induced by the planar embedding of G'', faces C_1 and C_2 must have opposite orientation). If C_1 and C_2 are

Figure 3.8: Routine to build the auxilliary graph

1. **function** `build_G''`$(graph : G; cycle : C; edgelist : E1, E2)$

   /* chordless C is a cycle of G */

   /* E_{at} for C in G is $E_1 \cup E_2$ */

   /* $E_1, E_2 \neq \emptyset$ */

   begin

2. copy G to G'' /* interpret C as C_1 */
3. append $|C|$ empty vertex lists to G'' /* for C_2 */
4. insert the $|C|$ edges to make C_2 a cycle
5. append 2 empty vertex lists v_1, v_2 to G'' /* hubs for C_1 and C_2 */
6. **for** $e \in E_2$

   delete $e = (c, v)$ from G'' /* c is the end in C_1 */

   insert $e = (c', v)$ in G'' /* c' in C_2 corresponds to c */
7. insert the the $|C|$ spoke edges (c, v_1) for c in C_1
8. insert the the $|C|$ spoke edges (c, v_1) for c in C_2
9. **return** G''

   end of `build_G''`

in different blocks of G' (or of G'') the problem is easily solved. If the distinct blocks of G'' are planar, we may simply reverse the orientation at every vertex in the block with one of the cycles (if they were similarly oriented), without affecting the (zero) genus. The cut vertex appearing in both blocks has its adjacency sublists for distinct blocks distinct.

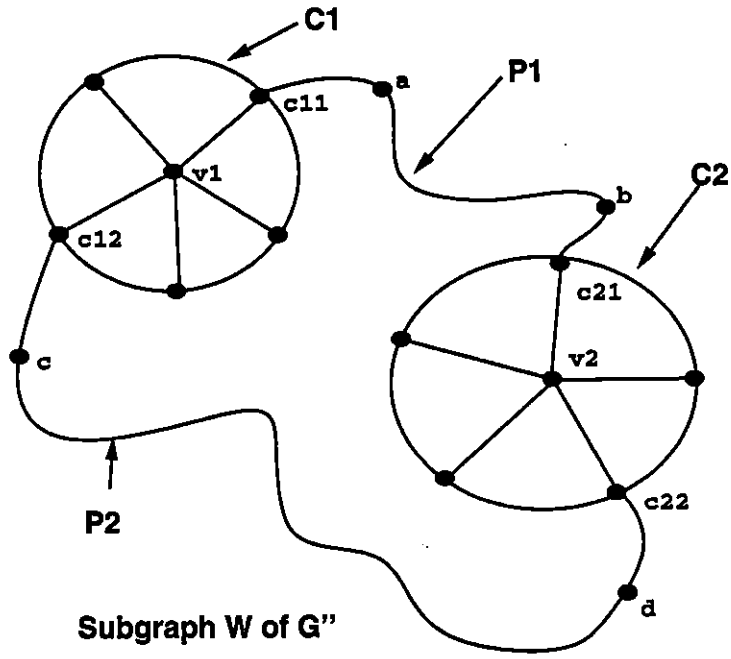
The interesting case is when G'' is a block. In this case we need a result of Menger:

Theorem 3.2.4 (Menger: See Gibbons [Gib85, p.109]) . *A graph is k-vertex connected if and only if any two distinct vertices are connected by at least k internally vertex disjoint paths.* □

Menger's theorem tells us that if G'' is a block (or if both cycles occur in the same block of G''), that there exist two vertex-disjoint (except for the end vertices) paths connecting the hubs v_1 and v_2. Finding two such paths can be done efficiently with a standard flow algorithm, also to be found in Gibbons [Gib85]. We call the routine to do this `find_2paths`. If G'' is not a block, `find_2paths` will find the articulation vertex.

The two paths found by `find_2paths` will connect a distinct pair of vertices c_{11} and c_{12} on C_1 with a distinct pair c_{21} and c_{22} on C_2, by paths $(v_1, c_{11}, a, \ldots, b, c_{21}, v_2)$, and $(v_1, c_{12}, c, \ldots, d, c_{22}, v_2)$ as shown in Figure 3.9. The graph W composed of the two wheels and the two paths, a subgraph of G'', is clearly planar, and can be embedded in the plane with either cycle having either orientation, and such that neither rim is in the interior (the side including the hub) of the other. Let Π be such a planar embedding of this subgraph in which the cycles have opposite orientation. We will call such an embedding of W *proper* and name the simple routine to do it `embed_2paths`. We may now state the main proposition supporting algorithm C.

Figure 3.9: Embedding two paths



Proposition 3.2.5 *Nonplanar block G has a toroidal embedding with chordless cycle C noncontractible if and only if there exists a nontrivial bipartition* (E_1, E_2) *of the edges of attachment of C to* $G - C$*, such that either 1. the auxilliary graph G'' has a planar embedding and the cycle copies* C_1 *and* C_2 *occur in different blocks of G'' or 2. G'' is a block and has a planar embedding, and any proper embedding of subgraph W is admissible (with respect to planar G'').*

Proof: If nonplanar block G has a toroidal embedding with cycle C noncontractible, then the planar embedding $G_{\Pi CC}$ of proposition 2.3.2 can be modified to the embedding Π needed here simply by inserting 1. a hub vertex inside each of the two faces and 2. the spoke edges. The definition of $G_{\Pi CC}$ and the proof of proposition 2.3.2 show that the induced embedding of G'' is such that neither E_1 nor E_2 is empty (or G would be planar), that neither C_i is embedded (in Π) in the interior of the other,

and that C_1 and C_2 have opposite orientations.

For the proof in the other direction, we have argued above that if G'' is not a block, (the cycles C_1 and C_2 occur in different blocks), that the embeddings of the block can be transformed to give the embedding required by the proposition. If however G'' is a block we suppose given a bipartition (E_1, E_2) with $E_1, E_2 \neq \emptyset$ and an arbitrary planar embedding Π of W such that neither C_i occurs inside the other, and such that the C_i are oppositely oriented (as faces of the appropriate subgraph). We are supposing (by hypothesis) also that Π is admissible relative to G'' which is planar. Then denote by Π' a planar embedding of G'' to which Π can be extended. By proposition 3.2.2 Π' can be transformed to a planar embedding Π'' of G'' in which the only component of $int(C_i)$ is v_i. Therefore in the induced (Π'') embedding of $G'' - \{v_1, v_2\}$, the C_i are both faces and have opposite orientation. Therefore a toroidal embedding of G can be constructed directly from Π''. □

3.2.5 Extending an embedding of W to one of G''

By proposition 3.2.5 a toroidal embedding of nonplanar block G can be found if a proper embedding of W is admissible relative to G''. But the planar embedding routine given early in this chapter iterates by extending admissible embeddings of a given graph, and is guaranteed to find a planar embedding of the given graph if any G_i is admissible. Therefore we can simply modify `planar` to get `demoucron` by letting G_1 be the proper embedding of W. Therefore, demoucron guarantees to find a planar embedding of G'' in which the proper embedding of W is admissible if such a planar embedding of G'' exists (and *not* just *if G'' is planar*). The time complexity of `demoucron` is clearly the same as that of `planar`.

The code for testing a given candidate cycle, with a given nontrivial bipartition

of edges, for being a noncontractible cycle in a toroidal embedding of the input graph G draws the theory of this section together in routine `test_partition`. Pseudocode for `test_partition` is given in Figure 3.10. Although routine `test_partition` returns only **true** or **false**, for success or failure, the actual toroidal embedding of the input graph is computed by `aux_to_tor` described in the next subsection, and deposited in Π at steps 9 and 17. Since `demoucron`, `find_2paths`, `build_G''`, `embed_2paths`, and `aux_to_tor` are all low degree polynomial in terms of time complexity, `test_partition` too is theoretically efficient.

3.2.6 Constructing a toroidal embedding from the auxiliary graph

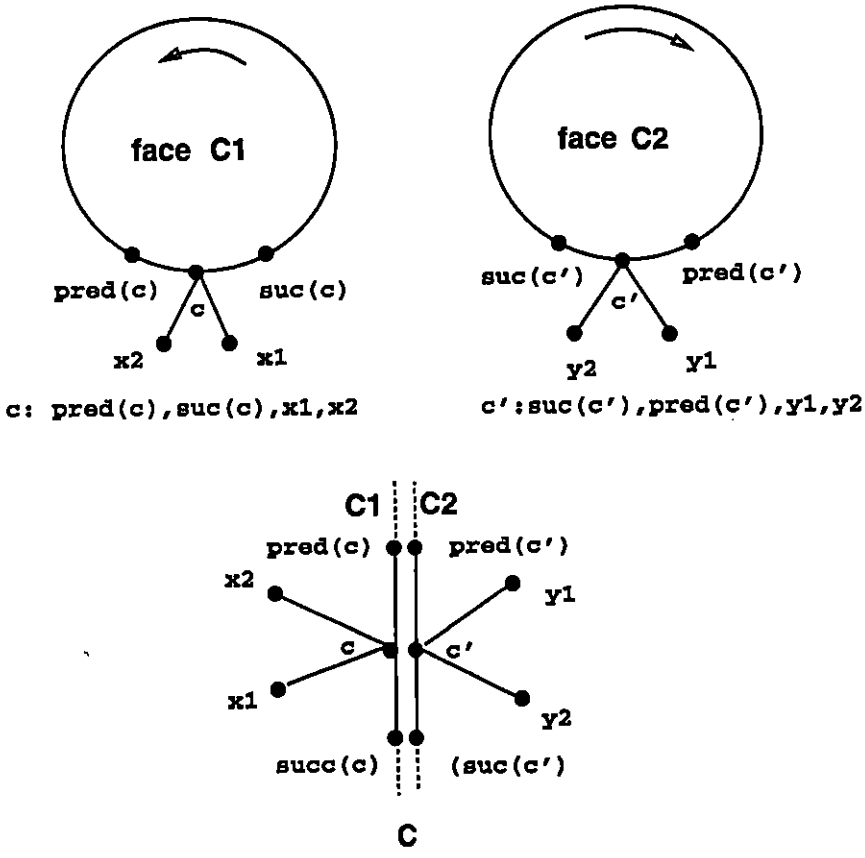
Routine `aux_to_tor` takes as input a graph (now interpreted as a *planar* embedding) of the sort created by `build_G`. The planar embedding is not just any planar embedding, but is such that C_1 and C_2 are oppositely oriented as is the case when they are in distinct blocks (so that one block can be inverted) or the embedding has been produced by `demoucron` as described in a later section. Routine `aux_to_tor` creates a toroidal embedding of G by reidentifying C_1 and C_2 after 2-switching the planar components (except for v_1 and v_2) from $int(C_1)$ to $ext(C_1)$ and from $int(C_2)$ to $ext(C_2)$, and deleting v_1 and v_2. The assumption that C_1 is oriented counterclockwise means that the vertices of C_1 as a face in the embedding induced when v_1 is deleted occur in the order of (oriented) C. The assumption that C_2 is oriented clockwise means that the vertices of C_2 as a face in the embedding induced when v_2 is deleted occur in the opposite order of (oriented) C. As long as C_1 and C_2 are oppositely oriented, this orientation can be achieved by a (possible) inversion of the orientation of the given embedding of G''. These orientation considerations are illustrated in

Figure 3.10: Routine to test a cycle with a given bipartition

```
1. function test_partition(nonplanar block : G; cycle : C; edgeset : E1, E2)
   /* (E1, E2) is a nontrivial bipartition of Eat for chordless C */
   /* v1 and v2 are the hub vertices of G'' */
   /* returns true if the sought embedding is found */
      begin
2.    G'' ← build_G''(G, C, E1, E2)
3.    find_2paths(v1,v2)
4.    if a cut vertex was found by find_2paths
5.       if planar(G'')
            begin
6.          Π ← planar embedding of G''
7.          if C1 and C2 similarly oriented in Π
8.             invert in Π the block with C2
9.          Π ← aux_to_tor(Π)
10.         return true
            end of "if this G'' planar", cut vertex case
11.      else return false /* G'' not planar, cut vertex case */
12.   else /* find_2paths found 2 paths */
         begin
13.      Π1 ← embed_2paths
14.      demoucron(Π1) /* Succeeds if Π1 is admissible in G'' */
15.      if demoucron failed return false
16.      else /* demoucron produced planar embedding Π */
17.         Π ← aux_to_tor(Π)
18.         return true
         end of 2 path case
      end of test_partition
```

Figure 3.11 where *pred*(*c*) and *suc*(*c*) are (resp.) the successor and predecessor of vertex c in oriented C, and C_1 and C_2 are shown as oriented faces in fragments of the embedding as required by the routine.

Figure 3.11: Glueing the two copies of the cycle back together again



3.3 Bases and families of cycles

Since we cannot directly produce the chordless noncontractible cycle required, nor the right partition of the associated set of edges which will result in a toroidal embedding of G if one exists, we are interested both in finding a small family of candidate

Figure 3.12: Routine to build the toroidal embedding

```
1. function aux_to_tor(embedding : Π″)
   /* Assumption: input G″ is a planar embedding */
   /* Assumption: int(C_i) consists of v_i, i = 1,2 */
   /* Assumption: C_1 oriented counterclockwise */
   /* Assumption: C_2 oriented clockwise */
   /* Constructs a toroidal embedding in Π (of G) */
      begin
2.    copy Π″ to Π
3.    for i = 1,2 do
         begin
4.       for each planar component (but not v_i) of int(C_i) do
            begin
5.          compute cycle C_xy bounding H in embedding Π″
               /* This cycle is determined by edge {x,y} of C_i */
               /* (in the induced embedding of H) */
               /* where x and y are the vertices of attachment of H */
               /* Caution: orientation different for C_1 and C_2 */
6.          flip(Π″, C_xy, x, y)
            end of planar component
         end of i = 1,2
7.    delete v_1 and v_2 and all their incident edges from Π
8.    for all c in C_1 /* c′ is the corresponding vertex in C_2 */
9.       insert after pred(c) and before suc(c) in the list of c,
            the sublist of c′ following pred(c′) and before suc(c′).
10.      delete c′ and its lists from G
      end of aux_to_tor
```

chordless cycles (guaranteed to include a noncontractible cycle if one exists – we will call this a *candidate family*), and in minimizing the associated set of edges of attachment, E_{at}, since we will have to try all bipartitions. This latter is particularly important for algorithm C because the number of bipartitions of a set is exponential in the size of the set. Therefore although a candidate family of cycles of size $O(1)$ can be found in linear time as described in Chapter 2, nevertheless, because of the complexity of the algorithm to do so, and especially because we have no control over the size of E_{at} we prefer another sort of family.

Horton [Hor87] shows how to find a shortest cycle basis (one with a fewest number of edges) in time $O(n_e^3 n_v)$, which for graphs of bounded genus translates to $O(n_v^4)$. Shorter cycles will tend to produce smaller size E_{at} of course, but an improvement is possible. As Horton points out, the algorithm does not suffer for attaching arbitrary positive weights to the edges. We want to attach to each *cycle* the weight equal to E_{at} for that cycle. The edge weighting function which accomplishes this is $w(u,v) = (d_u + d_v)/2$. Although the modified algorithm does not apply to graphs in which vertices of degree two or less occur, this is agreeable with our context, in which we reduce and eliminate small degree vertices before submitting a graph for toroidality testing. We call the routine to generate this basis `min_wt_basis`. Cycles of these bases are chordless.

3.4 Algorithm C

The pseudocode for algorithm C is given in Figure 3.13.

Although algorithm C bypasses the costly step of generating all planar embeddings of a graph, and shows an improvement in performance over the naive algorithm, the exponential bottleneck presented by the need to generate all bipartitions of edge set

Figure 3.13: Algorithm $\mathcal{C}$ to test for toroidality

```
1. function c_toroidal(nonplanarblock : G)
   /* v1 and v2 are the hub vertices of G'' */
      begin
2.    G' ←ser_par_red(G)
3.    if n_e > 3n_v (G') print "NONTOROIDAL"
4.    F ← min_wt_basis(G')
5.    for C ∈ F if planar(G' − C)
6.       for each nontrivial bipartition (E1,E2) of E_at for C
7.          if test_partition(G, C, E_1, E_2)
               print "TOROIDAL" and exit
8.    print "NONTOROIDAL" /* by exhaustion */
      end of c_toroidal
```

E_{at} for every candidate cycle tested, even though the cycle bases we use minimize the cardinality of E_{at}, manifests itself soon and forcefully in empirical tests.

In the following chapter we consider a toroidality testing algorithm which is much faster, possibly polynomial time, but which is not complete — that is, for some input graphs it returns "INCONCLUSIVE". Therefore it is logical to use the fast algorithm as a preprocessor for the complete algorithm.

The time complexity of `planar` is overshadowed by the exponential complexity of edge partitioning in the toroidality testing algorithm $\mathcal{C}$ that uses them both.

Chapter 4

Algorithm $\mathcal{D}$

4.1 Overview

4.1.1 Algorithm $\mathcal{D}$ and other algorithms in this thesis

Algorithm $\mathcal{D}$, the toroidality tester presented in this chapter is faster than $\mathcal{C}$, but sometimes returns "INCONCLUSIVE". It can be used as a preprocessor for $\mathcal{C}$. A more profound unification with $\mathcal{C}$ is possible, in which information accumulated in the preprocessor is passed to $\mathcal{C}$ where it continues to be accumulated and used. For clarity of presentation $\mathcal{D}$ is presented here first as an independent algorithm. Then in the following chapter, one way of unifying the algorithms that has been implemented with good effect is presented.

4.1.2 Algorithm $\mathcal{D}$ in outline

By way of motivation for the theoretical developments in this chapter, we now outline the basic method and technique of the algorithm, and present the pseudocode shell,

`d_toroidal`, in Figure 4.1. The most important routines called directly or indirectly by `d_toroidal` are given in pseudocode throughout the chapter. Step 5 is remarked upon in Section 4.3. As with algorithm $\mathcal{C}$, the input graph is a nonplanar block G with $n_e \leq 3n_v$, since by theorem 2.2.1 genera of blocks add, and by Proposition 2.2.2, if $n_e > 3n_v$, G cannot be toroidal. Step 4 is justified by the fact that K_7, the complete graph on 7 vertices, is toroidal (see Figure 4.2 for a toroidal embedding of K_7. Opposite sides of the rectangle are identified.)

The basic method is proof by contradiction. A nonplanar graph is assumed to be toroidal, and then a contradiction is produced, or a toroidal embedding is forced, or an answer of "Inconclusive" is returned. In contrast with algorithm $\mathcal{C}$, which returns "NONTOROIDAL" only after an exhaustive search for a toroidal embedding, algorithm $\mathcal{D}$ returns "TOROIDAL" only after an extensive search for incompatibilities with toroidality (a toroidal embedding is found however).

A basic technique is *densification* by *triangulation*: we add edges to G without destroying toroidality. We show how to do this in Section 4.2. We discuss in Section 4.3 the family of cycles needed as candidates for triangulation. We describe in Section 4.4 partial embeddings, the "data structure" used by routine `rpt` ("recursive partial triangulation", for which pseudocode is given in Section 4.6) to manage the information accumulated as we find cycles suitable for densification by triangulation. Routine `rpt` eventually, after repeated addition of edges to G, without destroying toroidality, reaches one of the following states:

1. $n_e > 3n_v$ which we call *edge overflow*, or one of several other easily detectable configurations incompatible with toroidality occurs. This is discussed in Section 4.5. We conclude G cannot be toroidal, and `rpt` passes this information back to `d_toroidal`, the calling routine.

2. $n_e = 3n_v$ and the method (that is, `rpt` itself) adds no more edges. There is a specialized algorithm (`rpt_tor`) for testing toroidality of this class of graphs. We discuss it in Section 4.7 and give pseudocode in Section 4.8.

3. $n_e < 3n_v$ and the method (`rpt`) adds no more edges. In this case `rpt` and algorithm $\mathcal{D}$ itself returns "INCONCLUSIVE". The specialized algorithm (`rpt_tor`) just mentioned can easily be modified to deal also with this case, and so $\mathcal{D}$ would be complete. But the cost (time) of using the specialized algorithm for this case is too great.

The complexity of `d_toroidal` is the complexity of `rpt` since the complexity of `rpt` is super linear.

Figure 4.1: Algorithm $\mathcal{D}$ (shell)

1. **function** `d_toroidal`(*graph* : G)
   /* Assumption: G is a nonplanar block */
   begin
2. $G' \leftarrow$ `ser_par_red`(G)
3. **if** $n_e > 3n_v$ **return** "NONTOROIDAL"
4. **if** $n_v < 8$ **return** "TOROIDAL"
5. $G'' \leftarrow G'$ with vertices ordered by increasing degree
6. **return** `rpt`(G'')
   end of `d_toroidal`

4.2 Densification by Triangulation

We show how to add edges to a nonplanar graph of unknown genus without affecting the toroidality of the graph. (If the graph was toroidal before the addition, it will be toroidal after the addition.)

4.2.1 Definitions

We call a cycle C of nonplanar G *flat* if C is contractible in *every* toroidal embedding of G. A *chordless* subgraph H of G is one which has no chords in G. A subgraph H *separates* G, or is a *separating* subgraph of G if $G - H$ consists of more than one component. Otherwise we say H is *nonseparating* in G. We say cycle C is a *flat face* of G if C is nonseparating, chordless, and flat. These conditions imply that C must be a face in every toroidal embedding of G.

A cycle C is *triangulated* by starting with a planar embedding of C and then adding edges (chords) to just one side of C until no more can be added An embedding is *triangulated* if every face is 3-sided. We say a *graph* is *triangulated relative to the torus* (or *fully triangulated* relative to the torus) if $n_e = 3n_v$. The justification for this latter terminology, is that if a graph G is triangulated relative to the torus, then any toroidal embedding of G will be a triangulation of the torus. Since we are only concerned in the present with toroidality, we omit to overtly add "relative to the torus" in the following. A graph is *densified* if we add edges, so that the ratio n_e/n_v approaches three (the limit value for toroidal graphs) without losing toroidality. We *densify* a graph by *triangulation* when we add edges to it by triangulating a cycle of it that we know must be a face in some toroidal embedding of the graph *if the graph is toroidal.* Thus, in particular, if G is nontoroidal, then G can be densified by

triangulating any and every cycle. The interesting context of course is that in which we are unsure of the genus of G.

4.2.2 Cycles which can be triangulated

A class of cycles that can be triangulated is specified by the following

Proposition 4.2.1 *If cycle C of toroidal graph G is such that $G - C$ is nonplanar then C is flat in G.*

Proof: Corollary to Proposition 2.3.1. □

The converse of this proposition (viz. cycle C of toroidal G is flat implies $G-C$ is nonplanar) is not true. This converse is not to be confused with the fact expressed in Chapter 2, Section 3, on Contractibility, for which K_7 is given as a counterexample. The sort of example that is easy to see as a counterexample is a large enough toroidal square grid in which a contractible cycle meanders through all or most of the graph as in Figure 4.3 in which C is outlined in bold, and opposite sides of the rectangle are identified to make a torus.

Proposition 4.2.2 *If cycle C is a flat face of nonplanar G then G' obtained from G by triangulating C is toroidal if G is.*

Proof: Suppose Π is a toroidal embedding of G. Since C is chordless and nonseparating, $int(C)$ is empty (by Proposition 2.3.1), and C itself is a face of Π. Therefore the Π' embedding of G' obtained by triangulating C is also toroidal, and therefore G' is toroidal. □

Figure 4.2: K_7 embedded in the torus

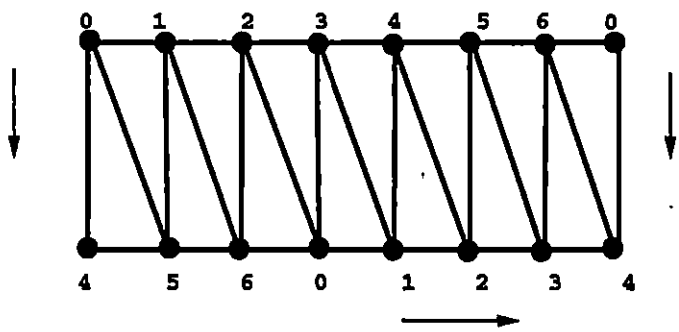
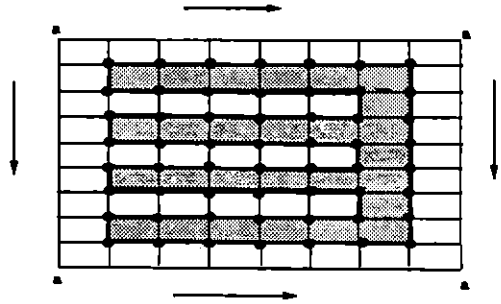


Figure 4.3: C is flat but $G - C$ is empty



Proposition 4.2.2 generalizes. It applies even if C is a face in *some* toroidal embedding. But the generalization is less useful (in this context) since it is not apparent how to recognize such cycles without first being given a toroidal embedding.

4.3 A family of candidate flat faces

The efficacy of `rpt` (the main routine called by `d_toroidal`, pseudocode given in Section 4.6, Figure 4.7), and the specialized algorithm `rpt_tor` (the routine to test toroidality of triangulated graphs, pseudocode given in Section 4.8, Figure 4.13) depends on being able to find a substantial list of flat faces quickly. Although it is not urgent here, as in algorithm $\mathcal{C}$, to find a minumum weight basis, the basic stock of cycles from which the minimum weight basis is chosen seems to be a good family of candidates for flat faces because they are chordless, they tend to be short. As a heuristic principle, flat faces too are chordless and tend to be short. Therefore we use here too, sets of fundamental cycles of breadth first search at each vertex, eliminating those with chords. We generate this set with routine `chordless_fam` whose time complexity, using breadth first search, eliminating cycles with chords, and storing as lists is $O(n_v^3)$. What is the best family of candidate cycles to use in searching for the flat faces is an open question.

Step 5 in `d_toroidal` 4.1 is not essential but it is observed that more flat cyles are found when the cycle search is carried out on a graph whose vertices are ordered by increasing degree. Aside from empirical evidence we offer for this strategy only the observation that a greater variety of corners is likely to be found by breadth first search *at each vertex* when a graph has this input format.

4.4 Partial embeddings: data structure

As we densify a graph, accumulating a list of its flat faces, if the original graph G was nontoroidal we may encounter various configurations incompatible with toroidality, thus settling the question for G. On the other hand we may arrive at a list of $n_e - n_v$ flat faces which either constitute a toroidal embedding or (by the fact that they do not constitute a toroidal embedding) prove nontoroidality. Possibly neither of these events develops. In every case we need an efficient structure for the accumulated data. A "partial embedding", soon to be defined, turns out to be a more useful structure than a simple list of flat faces.

4.4.1 Definitions

A *constraint* $\{a, b\}_v$ in the adjacency list of vertex v (interpreted as an embedding) is simply an adjacency of a and b in the list of v. We may abbreviate $\{a, b\}_v$ to $\{a, b\}$ if v is clear from the context. We define the *corner* of face $(\ldots, a, v, b, \ldots)$ at vertex v to be the (unoriented) 3-subsequence (a, v, b) of the face. A *compatible* set of constraints for a vertex v is a set that can appear in some embedding (of the graph in which v is a vertex). It is clearly possible to specify constraints for v that are *incompatible*: for example $\{\{x, a\}, \{x, b\}, \{x, c\}\}$. A peculiar special case arises when v has degree two—we want to say that a constraint occurs twice. But in our context, since the graphs have minimum degree three, we do not deal with the special case. The special case shows that, in the general case, constraints (like faces, edges, and embeddings themselves) must be considered as oriented. Note that in our application, we also consider faces, edges, and embeddings themselves, as nonoriented (we identify the objects of opposite orientation) except when we specify otherwise.

The following facts obtain for an embedding Π of a graph of minimum degree

three:

1. There is a natural one-one correspondence between the constraints of Π and the corners of the faces of Π.

2. Constraint $\{a,b\}_v$ corresponds to corner $\{a,v,b\}$ in one face of Π.

3. The degree d_v of a vertex v equals the number of constraints D_v in the list of v in Π, equals in turn, the number of corners incident to v in Π.

4. An embedding determines a set of d_v constraints for each vertex v of the embedded graph.

5. The set of constraints determined by Π correspond to just 2^{n_v-1} other embeddings because the set of constraints for a particular vertex can assume two orientations. For example if $\{\{a,b\},\{b,c\},\{a,c\}\}$ is a set of constraints for some vertex the list for that vertex could be (a,b,c) or (a,c,b).

4.4.2 Allowable sets of constraints

A list of flat faces determines a set of constraints: a flat face of k edges has k corners corresponding to k constraints. (But not every set of constraints determines a list of flat faces.) We denote a set of of compatible constraints by Π, the same symbol as for an embedding since it can be viewed as a *partial embedding* or as a generalization of an embedding. Or an embedding can be seen as a *full* set of constraints together with an orientation for each list. A set of constraints Π of graph G which can be extended to an embedding of G (by adding more constraints and assigning orientations) is *allowable.* A set of constraints which is allowable and can be extended to a *toroidal* embedding is *torus allowable.* A torus allowable set of constraints is analogous to an

admissible embedding of a subgraph of a planar graph. Analogous to the way in which the planarity algorithm builds an increasing series of admissible subgraphs, routines `rpt` and `rpt_tor` build an increasing series of torus allowable partial embeddings.

4.4.3 Data structure for partial embeddings

If the constraints corresponding to every flat face found for a graph G are entered into their respective lists (each vertex has such a list), then they align themselves like pieces of a linear jigsaw puzzle into (for each vertex) a list of ordered but unoriented sublists. (We treat this rigorously in the following.) For example, if v in G has list

$$v : 0, 3, 4, 6, 7, 8, 9$$

and partial embedding Π of G is such that

$$\Pi_v = \{0, 3\}, \{3, 6\}, \{4, 8\}, \{8, 9\}$$

then we can equivalently write

$$\Pi_v : (0, 3, 6), (4, 8, 9), (7)$$

If now constraint $\{0, 4\}$ is added to the partial embedding, the set of sublists of v becomes

$$v : (6, 3, 0, 4, 8, 9), (7)$$

It is necessary to generalize our notation for list Π_v to express a set of disjoint sublists, each of which is non-circular but could become circular (a complete set of constraints for a vertex corresponds to a circular list). This is easy to do if an arbitrary orientation is allowed to each sublist (so the orientation of any sublist can be arbitrarily inverted). A vertex x in a sublist of v, then has a successor $succ_v(x) = \Pi_v(x)$ which has the

value -1 if x is last in the sublist. Vertex x in a sublist of v has a predecessor $pred_v(x) = \Pi^{-1}(x)$ which has value -1 if x is first in the sublist.

It is possible to maintain a data structure for a partial embedding in such a way that inserting (or deleting) constraints can be done in constant time, by associating with every vertex x for every list, a node which specifies two adjacent vertices (possible values of -1 for one or both), and an *end* designating the other end of the sublist in case the given vertex x is the end of a sublist (when one of the adjacencies will have value -1), and by associating with every vertex v its degree and the number of constraints already in its lists. The three node values can be stored in three n_v by n_v arrays indexed by v and x, and the latter two values can be stored in two arrays of length n_v indexed by v. In the following presentation of the theory, and of the pseudocode (as well as in the implementation) we use a slightly less efficient representation, in which insertions take time $O(n_v)$, because the sublists are represented as ordered and so one of them might have to be inverted for a particular insert, as in the foregoing example. Specifically, square arrays `prev` and `next`, indexed by v and x give the value (-1 if none) of $pred_v(x)$ and $succ_v(x)$

4.5 Incompatible Configurations

By an *incompatible configuration* we mean any property of a graph inconsistent with that graph being toroidal. From the preceding section, it will be apparent that certain clashes with the data structure specified for partial embeddings will occur when the attempt is made to insert certain constraints. Another type of incompatibility occurs when a constraint could be inserted, but any embedding which is an extension of the resulting partial embedding would not be toroidal. In this section we consider some incompatible configurations of both types.

4.5.1 Excess edge ratio

The simplest configuration incompatible with toroidality is *excess edge ratio*, meaning $n_e > 3n_v$, already discussed in Chapter 2 and proved (a corollary of Proposition 2.2.2 to be an incompatible configuration. This type of incompatibility will occur sometimes when a flat face is triangulated. The set of corresponding constraints may be allowable, but is not torus allowable.

4.5.2 Face overflow

By Euler's Formula, Equation 1.1 with $g = 1$ we have $n_f = n_e - n_v$ for a toroidal embedding. Therefore, if we accumulate a list of flat faces which is longer than $n_e - n_v$ we can conclude the graph is nontoroidal. This is the same type of incompatibility as the last: the corresponding set of constraints may be allowable but is not torus allowable.

4.5.3 Edge Overuse

We call the configuration in which an edge occurs in three distinct faces of G, *edge overuse* and it clearly is another indication of nontoroidality. See Figure 4.5 (a). This follows from the definition of *face* in Chapter 1: The only two faces using edge $\{a, b\}$ are those determined by (a, b) and (b, a). It applies to embeddings of arbitrary genus. That is, any set of constraints corresponding to a set of faces in which an edge occurs more than twice is not allowable.

Edge overuse (of edge $\{a, b\}$) can be detected when the attempt is made to insert the edge for the third time as (without loss of generality) a constraint $\{b, x\}_a$ corresponding to a face $(\ldots, b, a, x, \ldots)$. Since the edge has already been entered twice,

as constraints $\{b, y\}_a$ and $\{b, z\}_a$, we will have $\Pi_a : \ldots(\ldots, y, b, z, \ldots)\ldots$. Entry b in the list will be found to have already a predecessor and a successor, and the insert attempt will fail.

4.5.4 Dwarf rim

A final configuration incompatible with toroidality involves a set of faces incident on a single vertex.

Before stating the fact in combinatorial terms, we note that it is closely related to the "thumbtack folk theorem" (so called by Gross and Tucker in [GT87], p.117) proved by Harary and Rosen in 1976 [HR76]. Refer to Figure 4.4. A "thumbtack" is a topological space homeomorphic to

$$\{(x, y, 0)|x^2 + y^2 \leq 1\} \cup \{(0, 0, z)|0 \leq z \leq 1\}$$

The theorem characterizes a 2-complex (in our terms, a graph G together with a select set of cycles of G specified as being faces) as not locally planar (in our terms, not admitting an embedding) if and only if it contains a thumbtack.

Combinatorially, a thumbtack, when it is not an "overused edge" (already discussed), manifests as a *dwarf rim*: that is, a cycle of size $< d_v$ in a set of constraints for vertex v. An example of a dwarf rim (and therefore of an incompatible set of constraints) for vertex v with adjacency list

$$v : 0, 3, 1, 5$$

would be

$$\{\{0,1\}, \{0,5\}, \{1,5\}\}$$

because although it has fewer than the degreee of v, $d_v = 4$ elements there is no ordered list of v with the first three constraints. Constraints $\{0,1\}$, $\{1,5\}$, and $\{0,5\}$

Figure 4.4: Thumbtack

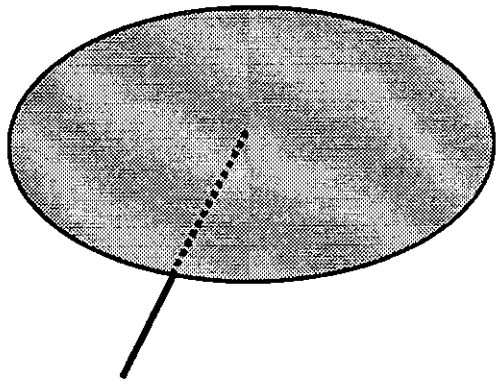
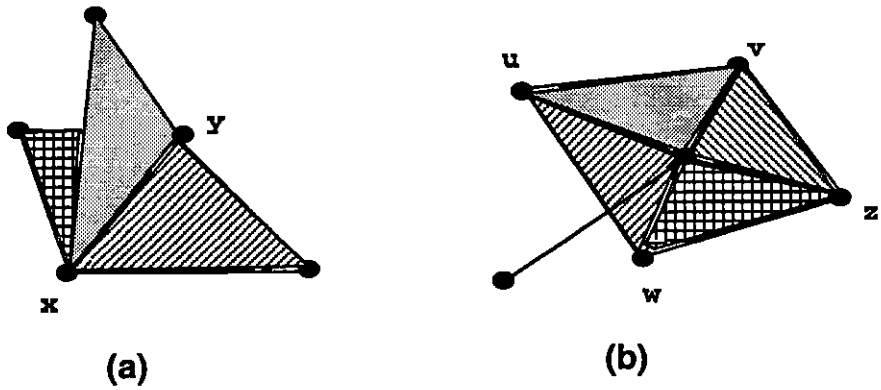


Figure 4.5: Combinatorial thumbtacks



enforce a circular sequence in *any* ordered list which includes them as constraints, and therefore, no list of constraints which includes them can include any other constraints. See the example in Figure 4.5 (b).

A set of constraints including a dwarf rim is not allowable, as the following proposition shows.

Proposition 4.5.1 *No embedding of a graph includes a dwarf rim.*

Proof: For proof of this perhaps obvious fact, we include the following consideration. If embedding Π of G included a dwarf rim $v_1, \ldots, v_k$, then there exists vertex w in the list of v but not in the vertices of the dwarf rim about v. Consider the finite sequence $w, \Pi_v(w), \Pi_v^2(w), \ldots, \Pi_v^{d_v-1}(w)$. Consider the entry z of the sequence just before the first entry which happens to be an element of the dwarf rim. Since π_v and the dwarf rim about v are both one-one and onto, $\Pi_v^{-1}(z)$ both is and is not an entry in the dwarf rim. This contradiction establishes the result. □

4.5.5 Routine to insert constraints in the partial embedding

Routine `insert_corner`, given in Figure 4.6, takes a corner (a, v, b) and inserts constraint $\{a, b\}$ in partial embedding lists Π_v, returning **true** or **false** in case of success or failure. Failure occurs when a dwarf rim or edge overuse would occur. Note that these are the only two configurations inhibiting the insertion of a constraint into a list that is not full. Thus the existence of this routine is another proof of the (combinatorial) thumbtack theorem.

Given the `next` and `prev` arrays as a data structure for partial embeddings, using $O(n^2)$ space and $O(n^2)$ setup time, the status of a and b in list of v in terms of succ and pred can be determined in constant time, but detecting whether they are in the

Figure 4.6: Routine to insert constraints in a partial embedding

```
1. function insert_corner(partial embedding : Π; vertex : a, v, b)
/* d_v is the degree of v */
/* D_v is the number of constraints already in Π_v */
    begin
2.  In lists of Π_v do
        begin
3.      if D_v = d_v return false
4.      if a has pred and succ already
            return false /* edge overuse */
5.      if b has pred and succ already
            return false /* edge overuse */
6.      if D_v < d_v − 1
            and a and b are in a common sublist
                return false /* dwarf rim */
7.      D_v ← D_v + 1  /* {a, b} will be inserted */
8.      if a has pred and b has pred
            invert sublist with b
9.      else if a has succ and b has succ
            invert sublist with a
10.     if a has succ and b has pred
11.         pred(a) ← b; succ(b) ← a
12.     else /* a has succ and b has pred */
13.         succ(a) ← b; pred(b) ← a
14.     return true
        end
    end of insert_corner
```

same subsequence (step 6) takes $O(n_v)$ time, as does inverting a sublist (steps 8 or 9). Updates of status and the insertion itself takes constant time. So the complexity of `insert_corner` is $O(n_v)$. As noted earlier, the operation can be reduced to constant time.

4.6 Recursive partial triangulation

4.6.1 The algorithm

The primary routine of algorithm $\mathcal{D}$ is `rpt`, which takes as input a block with no vertices of degree less than three, and with edge ratio not exceeding three. Pseudocode is given in Figure 4.7. After generating a family of candidate cycles (candidates for flat faces), each cycle is tested by routine `test_cycle` which inserts those cycles which are found to be flat faces in the list of flat faces, the corresponding constraints into partial embedding Π, and their triangulation edges into a list of triangulation edges which will be inserted into the graph only after all cycles are tested without contradiction to toroidality. Each insertion of a corner that succeeds leaves the partial embedding allowable (just by virtue of success of insertion) as well as torus allowable (since only corners of flat faces are being inserted into the constraint lists). Any failure of `insert_corner` thus implies that the current graph and so also the original graph are nontoroidal. So routine `test_cycle` returns false when `insert_corner` returns false and otherwise succeeds (returns true) when all `insert_corner` calls succeed (return true) (whether or not the cycle is a flat face).

The edges of triangulation (of the flat faces) are only inserted when every candidate cycle has been processed. (The flat faces are not refined, nor are the refined constraints added at this point because nothing is thus served.) And then `rpt` is called recursively.

Recursion is observed empirically never to go deeper than two levels. The number of recursive calls is bounded to $O(n_v)$ because at most $O(n_v)$ edges can be added before the graph is proved nontoroidal by edge overflow. When no more edges can be added and no contradiction to toroidality has been found and $n_e = 3n_v$, the specialized algorithm, `rpt_tor` is called, otherwise "Inconclusive" is returned.

The densified graph passed by `rpt` to `rpt_tor` is torus allowable with respect to the graph originally passed to the top level `rpt`.

4.6.2 Complexity of rpt

The time complexity of the called routine `rpt_tor` (not yet presented) will be denoted RT. Initialization of Π requires $O(n_v^2)$ time. Generating the family of $O(n_v^2)$ cycles takes time $O(n_v^3)$. Step 4 (`test_cycle`) is performed once for each of the $O(n_v^2)$ cycles. A graph can be analyzed for components, blocks and planarity in linear time. A cycle may require the insertion of $O(n_v)$ constraints at a cost of $O(n_v)$ each. However, no more than $O(n_v)$ constraints in all are inserted, and finding triangulation edges is linear in the length of the cycle, Therefore the cost of step 4 is $O(n_v^3)$. The cost of step 7, which is the cost RT of `rpt_tor` will be given in the analysis of that routine. In theory, the recursion in step 10 could occur $O(n)$ times so the entire routine costs $O(n_v^4) + RT$. But the recursion is observed empirically never to go deeper than two calls, and the complexity of `rpt` is conjectured to be $O(n_v^3) + RT$.

4.7 Toroidality of Triangulated Graphs

In this section we give the theoretical support for the specialized algorithm (`rpt_tor`) to determine the toroidality of triangulated graphs (those for which $n_e = 3n_v$).

Figure 4.7: The primary routine of algorithm $\mathcal{D}$

```
1. function rpt(graph : G)
   /* G is a nonplanar block with vertices ordered by increasing degree, */
   /* with n_e ≤ 3n_v, and no vertices of degree less than three. */
      begin
      /* Initialization of local data structures. */
2.    Π ← ∅ /* Partial embedding empty of constraints. */
      Face_list← ∅ /* Face_list of flat faces found. */
      Edge_list← ∅ /* Edge_list of triangulation edges. */
3.    Fam ← chordless_fam /* from bfs at each vertex */
4.    for all C ∈ Fam
          if not test_cycle(G, Π, C, Face_list, Edge_list)
              return "NONTOROIDAL"
5.    if (Edge_list= ∅) /* No triangulation edges to add. */
6.        if (n_e < 3n_v) return "INCONCLUSIVE"
7.        if (n_e = 3n_v) return rpt_tor(G, Π)
8.    else (Edge_list≠ ∅) /* There are edges to add. */
9.        GG ← G∪ Edge_list
10.       return rpt(GG)
      end of rpt
```

Figure 4.8: The routine to test for flat faces

```
1. function test_cycle(graph : G; partial embedding : Π; cycle : C;
                       list : Face_list, Edge_list)
   /* This routine is called only by rpt */
   /* Face_list is the list of flat cycles of G being built */
   /* Edge_list is the list of triangulaton edges of the elements of Face_list */
   /* returns false if contradiction to toroidality is found */
      begin
2.    if G − C has more than one nonplanar block
         return false
3.    if G − C has more than one component
         return true  /* Ignore C which separate G */
4.    if G − C has no nonplanar block
         return true  /* Ignore: unsure if C is flat */
5.    add cycle C to Face_list
      if |Face_list| > n_e − n_v return false
6.    add triangulation edges of C to Edge_list
7.    for all corners (a, v, b) of C
8.       insert_corner(Π, a, v, b)
9.       if insert failed return false
10.   return true
      end of test_cycle
```

4.7.1 Simplicial complexes

We note a comparison in the literature: A (2-dimensional) *simplicial complex* can be defined to be a graph together with a subset of its 3-cycles. A (2-dimensional) *simplicial 2-cell complex* can be defined to be a graph together with a subset of its cycles. A simplicial 2-cell complex is said to embed (in the torus) if both the graph embeds (in the torus) and every selected cycle embeds as a face in that embedding. The embeddability of simplicial 2-cell complexes has been explored by Harary, Gross, and Rosen [HR76, GR79, GR81]. Attributed to Filloti [GR81], is the observation that the genus of a locally planar 2-dimensional complex is computable by considering those rotation systems for the underlying graph that are consistent with the prescribed cycles of the complex. The implied algorithm is essentially the naive algorithm considered earlier except that constraints corresponding to the corners of the prescribed cycles limit the number of circular permutations to consider for each list. We could apply this algorithm to the 2-cell complex consisting of the graph together with the set of its known flat faces, as a substantial improvement over the naive algorithm. We find empirically that the cost (time) is still too great. However, in the case where $n_e = 3n_v$ the situation simplifies to allow a fast (but no proof yet of polynomial time) backtrack algorithm—`rpt_tor`.

4.7.2 Forced constraints

Let vertex v be of degree d_v in graph G. If $d_v - 1$ constraints about v have been inserted into the partial embedding Π, then Π_v has the form $(u_1, u_2, \ldots, u_{d_v})$ where this single list including every vertex adjacent to v is not oriented and *not circular*. One constraint remains to be inserted, and in fact can be inserted, since every embedding including the $d_v - 1$ constraints must also include the constraint $\{u_1, u_k\}$.

We translate and expand this into the fact we need in the following proposition. Note that although a simpler, "more powerful" theorem is possible, in that a constraint can be inserted even when the graph is not triangulated, the result is not interesting since as remarked above, processing graphs which are not triangulated using this method is too expensive (time).

Proposition 4.7.1 *If triangulated nonplanar graph G has torus allowable partial embedding Π, and there are $d_v - 1$ compatible constraints in Π_v and the remaining constraint at v is $\{x, y\}$, then partial embedding*

$$\Pi \cup \{\{x, y\} at\ v, \{v, x\} at\ y, \{v, y\} at\ x\}$$

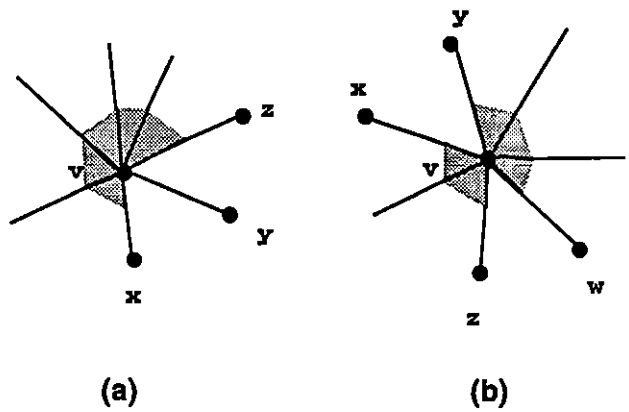
is torus allowable, and moreover, (x, v, y) is a face in every extension of Π to a toroidal embedding.

Proof: If G is toroidal, then since G is triangulated, every face in any toroidal embedding of G is 3-sided. Since there are $d_v - 1$ constraints in Π_v, the final constraint must occur in *any* embedding of G and so may be added to Π. This final constraint $\{x, y\}$ at v is the corner of a face which must be three sided and so can only be (x, y, v). Therefore any extension of Π to a toroidal embedding of G must include face (x, y, v) and so constraints $\{x, v\}$ at y and $\{y, v\}$ at x also. □

The preceding proposition shows how we can force three more constraints into a torus allowable partial embedding when Π_v is missing a single constraint (ie., has just $d_v - 1$ compatible constraints already). A stronger statement is possible: constraints can sometimes be forced even when Π_v is missing two constraints (ie., has just $d_v - 2$ constraints). In conjunction with the following proposition see Figure 4.9, in which constraints are shaded. The proposition applies to a set of constraints for a vertex v in which all existing constraints are contiguous (that is, compose a single sublist,

leaving the two uninserted constraints mutually adjacent) such as those of (a), but not to a set of constraints for a vertex v in which there are two nontrivial sublists (so that the uninserted constraints are not mutually adjacent) such as those of (b).

Figure 4.9: Forcing two constraints
Constraints are shaded



Proposition 4.7.2 *If triangulated nonplanar graph G has torus allowable partial embedding Π, and there are $d_v - 2$ compatible constraints in Π_v, and the remaining constraints at v are $\{x,y\}$, and $\{y,z\}$, (and NOT $\{x,y\}$ and $\{z,w\}$ with $x,y \neq z,w$), then partial embedding*

$$\Pi \cup \{\{x,y\}at\ v, \{x,v\}at\ y, \{y,v\}at\ x, \{y,z\}at\ v, \{y,v\}at\ z, \{z,v\}at\ y\}$$

is torus allowable, and moreover, (x,v,y) and (y,v,z) are faces in every extension of Π to an embedding of G in the torus.

Proof: Assuming G is toroidal, since G is triangulated, every face in any toroidal embedding of G is three sided. The two missing constraints in the list of Π_v are, by hypothesis, mutually adjacent, and therefore both must occur in *any* embedding which is an extension of Π. Therefore, as in the preceding proposition, two triangular

faces are forced to be in any toroidal extension of Π -viz., the faces together with their constraints listed in the statement of the proposition. □

4.7.3 Routine to insert forced constraints

Routine `force_constraints` although easily generalized to force constraints in arbitrary (as opposed to triangulated) partial embeddings is "streamlined" here for triangulated graphs. The routine must be passed an empty *Clist* because `rpt_tor`, the calling routine, does backtracking and so finds it necessary to remove forced constraints that follow an arbitrary constraint insertion. The routine `force_constraints` repeatedly traverses the full list of vertices of G until a pass has been made with no inserts (because an insert at one vertex will in general make possible forced inserts at another). Allowing a complete traversal per insert, since `insert_corner` has complexity $O(n_v)$, the worst case complexity for forcing is $O(n^2)$ per constraint, of which there are $O(n_v)$. But in view of the fact that with the appropriate data structures, constraint insertion takes constant time, this routine could be implemented in time $O(n_v)$.

4.7.4 Reification

An embedding Π of a graph G has d_v (the degree of v) constraints in the list for v for every vertex of G. We recall however that a set of d_v compatible constraints for each vertex v of G does NOT fully determine an embedding, because the orientation of any of the (circular because full) lists can be inverted. Inverting the orientation of one circular list certainly produces a different embedding. In fact a complete set of constraints (a full partial embedding) corresponds to 2^{n_v-1} different embeddings. If a graph is triangulated ($n_e = 3n_v$), however, just one of these exponentially many

Figure 4.10: Inserting the constraints forced at a vertex

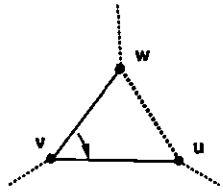
```
1. function force_constraints(graph : G; partialembedding : Π; list : Clist)
   /* Assumption: G underlying Π is triangulated. */
   /* return false if a contradiction to toroidality is found */
   /* the list of constraints inserted is returned in Clist (initially empty) */
      begin
2.    Done ← false
3.    while not Done /* repeat entire routine if any inserts were made last pass */
         begin
4.       Done ← true /* Done will be set false if inserts are made this pass */
5.       for all v ∈ V(G)
            begin
6.          if Π_v lacks just one constraint {a, b}
7.             if {a, b} ∉ E(G) return false
8.             insert the 3 corners of (v, a, b) in Π and in Clist
9.             return false if any insert failed
9a.            Done ← false
10.         else if Π_v lacks just two constraints {a, b}, {b, c}
11.            if {a, b}or{b, c} ∉ E(G) return false
12.            insert the 6 corners of (v, a, b) and (v, b, c) in Π and in Clist
13.            return false if any insert failed
13a.           Done ← false
            end of for all v ∈ V(G)
         end of while, a complete pass
      end of force_constraints
```

embeddings corresponding to a full set of constraints is a toroidal embedding. The following proposition states the essential fact. (Note that G, for example K_7 may have many distinct toroidal embeddings.)

Proposition 4.7.3 *If G is a triangulated block and if G has a toroidal embedding Π, then the orientation of every list of Π is determined up to a reflection of the embedding as a whole.*

Proof: Since G is a block, we need only show that given an orientation of one list of Π, say Π_v, the orientation of Π_w, for w in Π_v is determined (because orientation determination will then propagate through the block). So let w be in Π_v and $\Pi_v(w) = u$. Refer to Figure 4.11. Since Π is a triangulation of G, $\{w, u\} \in E(G)$ and so $u, v \in$

Figure 4.11: Reification



Π_w. The degree of w is at least 3, so at most one of $\Pi_w(u) = v$ and $\Pi_w(v) = u$ may hold. The former certainly holds or the face determined by (w, v) would have more than 3 edges, a contradiction. So the orientation of row w is uniquely determined, and the result follows. □

We call *reification*, the process of inverting orientations of some of the rows of a fully constrained partial embedding, so that the resulting embedding will be a triangulation of the torus. In Figure 4.12 we give pseudocode for reifying a given fully constrained partial embedding of a triangulated graph to produce a toroidal embedding if possible. The routine uses breadth first search to traverse the vertices of a

graph, inverting each list found incompatible with the current embedding interpreted as toroidal according to Proposition 4.7.3.

The complexity (for toroidal graphs) of breadth first search is $O(n_v)$ but also $O(n_v)$ lists of size $O(n_v)$ may need to be reversed, so the complexity of `reify` is $O(n_v^2)$. The routine could be implemented in time $O(n_v)$ by modifying the data structures so that lists do not have to be physically inverted but rather boolean coded for orientation.

4.8 Embedding triangulated graphs

Whereas in `rpt` a torus allowable partial embedding is built by inserting a constraint for each corner of each flat face found in a cycle family, `rpt_tor` inserts constraints which are implied by already existing constraints, as allowed by Propositions 4.7.1 and 4.7.2 of the current chapter, and when no more can be thus forced, inserts them arbitrarily, withdrawing them in case the tentative inserts lead to nontoroidal embeddings. And so by exhaustive backtracking, a toroidal embedding is found, or the graph is proved nontoroidal.

Theoretically, in view of the backtracking, the time complexity of `rpt_tor` could be exponential in n_v. The computational evidence shows that average case and possibly even worst case may not be so bad. The entire routine can of course be dropped from the algorithm, leaving an order n_v^3 algorithm of exactly the same type, but returning "INCONCLUSIVE" more often. Experimental evidence is that even in the unified algorithm described in the following chapter, it is preferable to retain routine `rpt_tor`.

Figure 4.12: Routine to reify a fully constrained partial embedding of a triangulated graph

```
1. function reify(partialembedding : Π)
   /* Assumption: Every vertex of G has minimum degree 3 */
   /* Assumption: G is triangulated */
   /* Assumption: Π is fully constrained */
      begin
      /* Build reified embedding in Π' */
2.    for all v in G Done(v) ← false
3.    Π' ← Π
4.    Done(0) ← true /* wlog Assume Π_0 is correctly oriented */
5.    add 0 to queue
6.    while queue not empty
         begin
7.       remove a vertex from queue, call it v
8.       for all w in Π'_v if not Done(w)
            begin
9.          if Π'_w(v) = Π'_v(w) then invert Π'_w
10.         Done(w) ← true
11.         add w to queue
            end of for all w
         end of while
      end of reify
```

Figure 4.13: Embedding a triangulated graph

```
1. function rpt_tor(graph : G; partialembedding : Π)
   /* Assumption: Π is torus allowable relative to triangulated G */
   /* rpt_tor is called once by rpt and recursively by rpt_tor */
      begin
2.    ListC ← ∅ /* List of locally inserted constraints */
3.    force_constraints(G, Π, ListC)
4.    if force_constraints failed
          remove the constraints of ListC from Π
          return "NONTOROIDAL"
5.    if Π is full /* ie. if Π now has 2n_e constraints */
6.        reify(Π)
7.        if (check_genus(Π) == 1) return "TOROIDAL"
8.        else return "NONTOROIDAL"
9.    else /* Π is not full and no forcing is possible */
          begin /* arbitrary constraint insertion and backtrack is necessary */
10.       find a vertex v ∈ V(G) with a minimum number of missing constraints
11.       choose a vertex w at an end of a sublist of Π_v
12.       for all constraints {w, x} possible to insert in Π_v
              begin
13.           if edge {w, x} ∉ E(G) continue /* try next constraint */
14.           insert the three constraints of (v, w, x) in Π
15.           if rpt_tor(G, Π) ="TOROIDAL" return "TOROIDAL"
16.           else remove the three constraints and continue
              end of for all constraints {w, x}
          end /* arbitrary constraint insertion and backtrack */
17.       remove the constraints of ListC from Π
18.       return "NONTOROIDAL"
      end of rpt_tor
```

Chapter 5

The Unified Algorithm

5.1 Overview

Because algorithm $\mathcal{D}$ is fast but not complete, and algorithm $\mathcal{C}$ is complete but exponential at best, an obvious improvement is to run the two algorithms in tandem. But more is possible. We describe in this section how the routines of $\mathcal{E}$ interface. In Section 2 we discuss modifications to algorithm $\mathcal{D}$, whose implementation in $\mathcal{E}$ is called `rpte`. And in Section 3 we discuss modifications to $\mathcal{C}$, whose implementation in $\mathcal{E}$ is called `ftor`. The modifications are profitable to introduce when $\mathcal{D}$ is run as a preprocessor for $\mathcal{C}$.

The unified program has a top level shell `e_toroidal` which is essentially identical to `d_toroidal` so we do not present it in a separate figure. The one detail to modify is that `e_toroidal` in line 6 calls `rpte` rather than `rpt`.

Routine `ftor`, is called once by `rpte`, at the point when "INCONCLUSIVE" would otherwise be returned.

5.2 The modified preprocessor algorithm $\mathcal{D}$

5.2.1 Overview

We list the criteria by which we decide what graph G, and when, should be passed by the preprocessor to the modified $\mathcal{C}$:

1. The preprocessor should maximize the number and proportion of flat faces known for G. This implies maximizing the number of edges, preferably to $3n_v$ or more. Modified $\mathcal{D}$ will continue to add to the list of flat faces.

2. The preprocessor should, roughly speaking, minimize the size of the graph passed to $\mathcal{C}$ in view of the edge partition bottleneck.

The version we present here is not the only possible compromise. Routine `rpte` while processing cycles, will *reduce* (see below) the graph at the earliest opportunity, recursing to the top level toroidality tester `e_toroidal` with a graph with a reduced number of vertices (as well as a reduced number of edges). But when `ftor` is finally called by `rpte` it will be with a graph that has been densified with as many edges as possible. That is to say, when `rpte` cannot reduce the graph nor does the graph fully triangulate, so that `rpt_tor` is not applicable, then the graph passed to `ftor` is the densified graph with a reduced number of vertices, rather than the graph as input to `rpte`.

5.2.2 Reduction

We say that graph G *reduces* to subgraph H when $|V(H)| < |V(G)|$ and H is toroidal iff G is. (So that we may determine the toroidality of G by determining the toroidality

of H.) The reduction as implemented in `rpte` is supported by the following proposition.

Proposition 5.2.1 *If C is a chordless cycle of nonplanar G, and $G - C$ consists of two disjoint components: planar P and nonplanar NP, and there exists a planar embedding of $G - NP$, then if $G - P$ is toroidal, so is G.*

Proof Any planar embedding of $G-NP$ may without loss of generality be considered such as to embed P in the interior of C (component P is not separated by C). But since NP is nonplanar and $G - P$ is toroidal, by proposition 2.3.1, $NP \subset ext(C)$ in any toroidal embedding of $G - P$, and so C is a flat face of $G - P$. But then any planar embedding of $G-NP$ may be combined with any toroidal embedding of $G-P$ (possibly inverting the orientation of one of the two), to get a toroidal embedding of G, by identifying the C in $G - NP$ with the C in $G - P$. □

5.2.3 Modifications to $\mathcal{D}$ for Unification

We modify the pseudocode for `rpt` and `test_cycle` in Chapter 4 to get `rpte` and `test_cyclee`. There are two extra parameters for `test_cyclee`: a boolean flag *reduce* which is set to **true** when a cycle has been found which allows reduction, and the graph G' to use for reduction. `test_cycle` itself still returns **false** only when a contradiction to toroidality has been found.

The only changes to `rpt` are

1. At step 5 (`rpte`), `test_cyclee` is called rather than `test_cycle`.
2. Reduction and recursion to `e_toroidal` occurs just after the call of `test_cyclee` if the *reduce* flag was set (by `test_cyclee`).

3. At step 8 (`rpte`), `ftor` is now called rather than returning "INCONCLUSIVE".

5.3 Modified algorithm $\mathcal{C}$

5.3.1 Overview

Algorithm $\mathcal{C}$ modified to take advantage of the work done by algorithm $\mathcal{D}$ as a preprocessor, is implemented as routine `ftor`, for which the pseudo-code is given near the end of this section. Algorithm $\mathcal{C}$ implemented as `ftor` is passed not only a nonplanar block to test for toroidality, but also the list of flat faces of that graph found by `rpte` and the corresponding partial embedding Π. The justifying proposition is given in the next subsection. The idea is to continue adding to the list of flat faces found by `rpte`, some of the cycles tested and rejected by algorithm $\mathcal{C}$. Not just any rejected cycle can be added to the list. But if chordless nonseparating cycle C is found by $\mathcal{C}$ not to be a noncontractible cycle (in any toroidal embedding of the graph) then C must be a flat face of the graph, and the partial embedding begun by `rpte` can be augmented.

5.3.2 More Flat Faces

The definition of flat face has been given as, a chordless, nonseparating cycle that is contractible in every toroidal embedding of the graph. Proposition 2.3.2 showed how to recognize a subclass of the flat cycles of a graph. The following proposition shows how to recognize another subclass of flat faces.

Figure 5.1: Modified preprocessor $\mathcal{D}$

```
1. function rpte(graph : G)
   /* G is a nonplanar block with vertices ordered by increasing degree, */
   /* with n_e ≤ 3n_v, and no vertices of degree less than three. */
      begin
      /* Initialization of local data structures. */
2.    Π ← ∅ /* Partial embedding empty of constraints */
      Face_list← ∅ /* Face_list of flat faces found */
      Edge_list← ∅ /* Edge_list of triangulation edges */
3.    Fam ← chordless_fam /* from bfs at each vertex * /
4.    for all C ∈ Fam
         begin
5.       if not test_cyclee(G, Π, C, Face_list, Edge_list, reduce, G')
            return "NONTOROIDAL"
6.       if reduce then return e_toroidal(G')
         end
7.    if (Edge_list= ∅) /* No triangulation edges to add. */
8.       if (n_e < 3n_v) return ftor(G, Pi, Face_list)
9.       if (n_e = 3n_v) return rpt_tor(G, Pi)
10.   else (Edge_list≠ ∅) /* There are edges to add. */
11.      GG ← G∪Edge_list
12.      return rpte(GG)
      end of rpte
```

Figure 5.2: Modified routine to test for flat faces

```
1. function test_cyclee(graph : G; partial embedding : Π; cycle : C;
                        list : Face_list, Edge_list; boolean : reduce; graph : G')
   /* returns false when contradiction to toroidality is found */
      begin
2.    reduce ← false /* default flag to not reduce and recurse */
3.    if (G − C has more than 2 components) return true /* ignoring this cycle */
4.    else if G − C has just 2 components then
         begin
5.       if (2 blocks are nonplanar) return false
6.       else if (both components are planar) return true /* ignoring this cycle */
7.       else /* just one of the 2 components is non planar */
                G' ← (G− planar component)
                reduce ← true
                return true
         end of case where G − C has just 2 components
8.    else begin /* G − C has just 1 component */
9.       if G − C has more than one nonplanar block
            return false
10.      if G − C has no nonplanar block
            return true /* Ignore: unsure if C is flat */
11.      add cycle C to Face_list
            if |Face_list| > n_e − n_v return false
12.      add triangulation edges of C to Edge\_list
13.      for all corners (a, v, b) of C
14.         if not insert_corner(Π, a, v, b) return false
15.      return true
      end of case and of test_cyclee
```

Proposition 5.3.1 *If C is a chordless cycle of nonplanar G, and $G - C$ is a single planar component, and for no nontrivial bipartition $(E1, E2)$ of the edges of attachment of C to $G - C$ does there exist a planar embedding of the auxilliary graph $G(C, E1, E2)$ in which the copies of C are oppositely oriented, then C is a flat face of G.*

Proof It follows from proposition 3.2.1 that C is contractible in every toroidal embedding of G. Since C is also nonseparating, it is a flat face of G. □

5.3.3 Modifications to $\mathcal{C}$ for Unification

We give the pseudo code for modified $\mathcal{C}$ as routine `ftor` below. It is called once from routine `rpte` (see Section 1). Routine `ftor` is a modification of routine `c_toroidal` of Chapter 3.

The theoretical time complexity of the unified algorithm is not different from that of algorithm $\mathcal{C}$. Testing even a single cycle in `ftor` can make the process exponential in n_v.

We note the following possible variant approach. Instead of generating a minimum weight cycle basis in `ftor` and skipping over cycles already known to be flat, we could instead generate a modified minimum weight basis starting with the list of faces known to be flat when `ftor` is called. In either case no cycles known already to be flat are processed by an exponential routine, and cycles tested by that routine and rejected, are added to the flat face list.

Figure 5.3: Modified C for unified algorithm

```
1. function ftor(nonplanar block : G; partial embedding : Π; list : Face_list)
   /* Called once from rpte. */
      begin
2.    F ← min_wt_basis(G)
3.    for C ∈ F if planar(G − C) and C ∉ Face_list
         begin
4.       for all nontrivial bipartitions (E1, E2) of Eat for C
            begin
5.          if (test_partition(G, C, E1, E2)) return "TOROIDAL"
6.          else if G − C consists of a single component /* C is a flat face */
               begin
7.             add C to Face_list
8.             insert corners of C in Π
                  returning "NONTOROIDAL" if any contradictions found
9.             if (nf = ne − nv)
10.               Π' ← reify(Π)
11.               if (check_genus(Π') = 1) return "TOROIDAL"
12.               else return "NONTOROIDAL"
               end of case of single component
            end of for all bipartitions
         end of for C
13.   return "NONTOROIDAL" /* by exhaustion */
      end of ftor
```

Chapter 6

Computational Evidence

6.1 Introduction

In this chapter we consider program correctness and performance based on computational evidence. It is hard to be sure of the correctness of a hundred pages of C code. If we are testing for toroidality, and a toroidal embedding is found, a simple check is possible, viz., verify using the face counting algorithm that the embedding is toroidal. But if the output is "NONTOROIDAL", the correctness is much harder to check. For illustration, an early version of the program was returning correct results for small graphs, but an error in assigning array sizes resulted in faulty output of the minimum weight cycle basis routine. The family of cycles output by the routine was not always a basis as specified, and so in some cases included no noncontractible cycle of the toroidal graph being tested. As a consequence a wrong output of "NONTOROIDAL" occurred after exhaustive search of the cycle family. Confirming the correctness of a "NONTOROIDAL" output is hardly easier when the result is obtained by contradiction as in algorithm $\mathcal{D}$, especially when backtrack is involved

as in routine `rpt_tor`, as must occur in some cases. Confirming the correctness of a "NONTOROIDAL" output is certainly no easier when the result is obtained by contradiction by algorithm $\mathcal{C}$ based only partly on information passed on by $\mathcal{D}$ (as opposed to proof by failure of systematic exhaustive effort to embed in the torus).

In order to assess empirically the correctness and performance of the program and variations of it, we executed a systematic search for the set of minor order obstructions and irreducible graphs (definitions below) on eight, nine, and ten vertices. The hope is to compare these results with those of other workers, and of other toroidality testers. Also, graphs which are obstructions present many closely related borderline cases, (presumably the hardest to test), and allow some intensive checking "by hand". Also, the list of obstructions could help some day to classify them all, which could give insight into a practical torus embedding algorithm as well as having theoretical interest.

By *obstruction* we mean either an irreducible graph (definition following section) without degree two vertices (these could be called "topological obstructions") or a minor order obstruction (definition following section).

6.2 Kuratowski's Theorem and Obstructions

We recall (from Chapter two) that according to Kuratowski's theorem, that there exist just two "obstructions" to planarity: viz., K_5, the complete graph on five vertices, and $K_{3,3}$, the complete bipartite graph on two sets each of three vertices. They are obstructions in the sense that any nonplanar graph can suffer deletions of edges and vertices and/or contractions of edges while maintaining nonplanarity, except these two. If a graph of genus k is such that any proper subgraph has genus less than k, then it is called (similarly to Archdeacon and Huneke in [AH89]) *irreducible.* Thus

for example there are two infinite families of graphs irreducible for the plane, but just two graphs with no degree two vertices irreducible for the plane. We also call irreducible graphs with no degree two vertices *topological obstructions.* Topological obstructions are "topological order obstructions" in Fellows and Langston [FL88].

It is possible for a graph that is a topological obstruction to be such that the contraction of some edge does *not* reduce the genus of the graph (in distinction to the deletion of any edge, which *does* reduce the genus). That is, a graph which is a topological obstruction is still sometimes "reducible" in the sense that it has an edge which contracted leaves the graph with the same genus. If a topological obstruction G also satisfies the property that the contraction of any edge leaves the graph with lower genus, then we say (with, for example, Fellows and Langston [FL88]) that G is also a *minor order obstruction.* This distinction is not effective for obstructions to planarity. That is, both topological obstructions to planarity are also minor order obstructions to planarity.

It was conjectured by Wagner in [Wag37], and proved by Robertson and Seymour in [RP], that the number of minor order obstructions to toroidality is finite. Our performance project involves generating the approximately 15 million isomorphically distinct connected graphs on eight to ten vertices and testing each. We discuss generating these graphs in the next section, and give the pseudo code in the following section.

6.3 Generating Isomorphically Distinct Graphs

Two graphs G_1 and G_2, (whose vertices are labelled), are said to be *isomorphic* if there exists a one to one mapping Φ of the vertices of G_1 onto the vertices of G_2 such that adjacency is preserved, viz., such that $\{\Phi(a), \Phi(b)\} \in E(G_2)$ if and only

if $\{a, b\} \in E(G_1)$. Clearly, genus is preserved by isomorphism. The number of isomorphically distinct graphs of a given species is evidently less, in general, than the number of labelled graphs of that species (in particular of genus). In order to expedite a systematic search for obstructions we need either lists or fast generators of the sets of isomorphically distinct graphs.

As inferred in Chapter one, isomorphism testing is not known to be an easy problem. However, for small enough graphs (no more than 16 vertices) Brendan McKay of Australian National University has developed a generator [McK90] that is highly optimized and effectively performs as well as reading from a list. It generates isomorphically distinct graphs at the rate of several hundred thousand per hour (in our range of interest). The generator is one component of McKay's **nauty** package of graph processing programs. We call it as `makeg` with several command line arguments (on a *unix* machine), as in the following example:

```
makeg -c -n 13 20 40 1000 333
```

The `-c` flag restricts output to connected graphs, the `-n` flag specifies output in *nauty* format (not relevant here), '13' specifies 13 vertices. (Generate graphs with 13 vertices.) '20' and '40' specify the minimum and maximum number of edges. '1000' specifies that just one of 1000 roughly equal sized disjoint classes of all isomorphically distinct 13 vertex graphs on 20 to 40 edges will be generated. '333' specifies which of the 1000 'modular' classes will be output.

Output from `makeg` is piped to the obstruction searcher. Using `makeg`, we have found that generating nonisomorphic graphs consumes less than one percent of processing time of our obstruction searcher.

6.4 The Obstruction Searcher

6.4.1 Additivity and Trivial Obstructions

Because genus of blocks is additive, we can dispense with testing non-blocks, as demonstrated in the following proposition, and since neither topological obstructions nor minor order obstructions have degree two vertices, we can dispense with testing any graph with a degree two vertex.

Proposition 6.4.1 *If G is a disconnected or has an articulation vertex which is a minor order (resp. topological) obstruction to toroidality, then G consists of two blocks each of which is a minor order (resp. topological) obstruction to planarity.*

Proof: The genus of G is two because G, being an obstruction to toroidality, is not toroidal, and deleting any edge yields a toroidal graph.

Case 1: The maximum genus of a block is two, Then the genus of every other block must be 0, because the genus of blocks is additive. But then edges (or isolated vertices) of the genus 0 blocks may be deleted while G maintains genus 2, contradicting the hypothesis that G is an obstruction. Therefore this case does not arise.

Case 2: The maximum genus of a block is 1. Then just two of the blocks have genus 1, or by the additivity thm, the genus of G would be greater than 2, a contradiction. So every other block has genus 0. But if there are blocks with genus 0, G is not an obstruction, since edges can be deleted without changing the genus of G. Therefore G consists of exactly 2 blocks, each of genus 1. If any edge of one block is deleted or contracted, (resp. deleted) the resultant G has genus one, so the suffering resultant block has genus 0. That is the suffering block is a minor order (resp.topological) obstruction to planarity. This holds for either block. Therefore the two blocks of the

obstruction G to toroidality are both minor order (resp. topological) obstructions to planarity. □

Since every minor order obstruction is a topological obstruction we restrict our search to a search for blocks that are topological obstructions, and when we find one, we test it to see if it is also a minor order obstruction. Thus our tables of obstructions do not include the six "trivial" two block obstructions. (There are six ways of combining the two obstructions to planarity.)

6.4.2 Pseudocode for Finding Obstructions

The top level routine `find_obstruc` for finding obstructions uses routine `e_toroidal` referred to in Chapter five as the top level toroidality tester implementing the combined algorithm for toroidality testing. The two routines `obstruc_minor` and `obstruct_topol` called are presented in pseudocode following the top level `find_obstruc`.

Following are the two routines called for nontoroidal graphs to test for being obstructions.

A unix command line to find all 9 vertex obstructions that are blocks and have 20 to 23 edges might appear as

```
makeg -c -n 9 20 23 | find_obstruc > outputfile
```

6.5 Tables for 8, 9, 10, and 11 Vertex Graphs

The tables in each subsection give distributions of graph obstructions by number of edges.

Figure 6.1: Routine to find obstructions

```
1. function find_obstruc(graph : G)
  /* Piped input, a list of isomorphically distinct connected graphs. */
     begin
2.   while graphs remain at input
        begin
3.      read next graph G
4.      if G has a degree 1 or 2 vertex continue /* ignore it */
5.      if G is not a block continue
6.      if G is planar continue
7.      if G is toroidal continue
8.      /* At this point G is a nontoroidal block with vertices of degree at least 3 */
9.      if obstruc_topol(G)
            begin
10.         print "G is a topological obstruction."
11.         if obstruc_minor(G)
12.             print "G is a minor obstruction."
13.         print G
            end of "if obstruc_topol"
        end of while loop
     end of find_obstruc
```

Figure 6.2: Routine to check G for being a topological obstruction

```
1. function obstruc_topol(graph : G)
  /* G input is a nontoroidal block with vertices of degree at least 3. */
     begin
2.   for all edges e of G
        begin
3.      delete e from G
4.      if G is not toroidal return false
5.      insert e in G
        end of for all edges
6.   return true /* G − e is toroidal for every edge e */
     end of obstruc_topol
```

Figure 6.3: Routine to check G for being a minor order obstruction

```
1. function obstruc_minor(graph : G)
  /* G input is a block which is a topological obstruction */
      begin
2.    for all edges e of G
          begin
3.        G' ← G with e contracted to a vertex
4.        if G' is not toroidal return false
          end of for all edges
5.    return true /* G.contract.e is toroidal for every edge e */
      end of obstruc_minor
```

6.5.1 Obstructions on 8 vertices

Since K_7 is toroidal, any obstruction to toroidality has at least 8 vertices. As shown in the table of Figure 6.5.1 we find just three graphs on 8 vertices which are (each, both topological and minor order) obstructions. An 8 vertex obstruction must have at least 12 edges because the minimum vertex degree is 3; and no more than 25 edges because every single edge deletion from an 8 vertex graph with more than 25 vertices must be nontoroidal by excess edge ratio. There are altogether 11,001 isomorphically distinct connected 8 vertex graphs on 9 to 25 edges. They take several minutes of Sparc station SLC time to process. The results are are summarized in the table of Figure 6.5.1 and the three obstructions are given in adjacency list form and sketched in Figure 6.5, where arguments for the nontoroidality of each is also included.

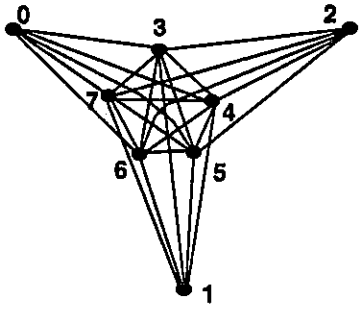
Figure 6.4: Obstructions to toroidality—8 vertices

number of edges	19	20	21	22	23	24	25		
number of obstructions	0	0	0	1	0	1	1	TOTAL	3

6.5.2 Obstructions on 9 vertices

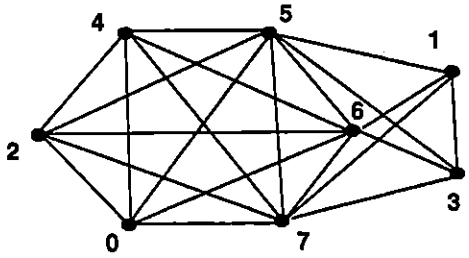
As shown in the table of Figure 6.6 we find just 47 graphs on 9 vertices which are topological obstructions. Of these, 42 are minor order obstructions. A 9 vertex obstruction must have at least 14 edges because the minimum vertex degree is 3; and no more than 28 edges because every single edge deletion from an 9 vertex graph with more than 28 vertices must be nontoroidal by excess edge ratio. There are altogether 260,536 isomorphically distinct connected 9 vertex graphs on 10 to 28 edges. Processing these graphs takes about ten hours on a Sparc station IPC.

Figure 6.5: The eight vertex obstructions



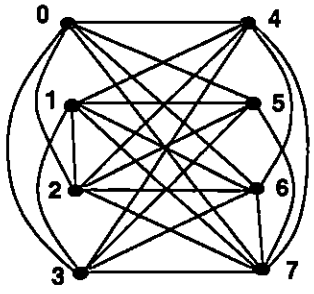
0: 3,4,5,6,7
1: 3,4,5,6,7
2: 3,4,5,6,7
3: 0,1,2,4,5,6,7
4: 0,1,2,3,5,6,7
5: 0,1,2,3,4,6,7
6: 0,1,2,3,4,5,7
7: 0,1,2,3,4,5,6

(Edge:vertex ratio of 25:8 > 3 means edge overflow)



0: 2,4,5,6,7
1: 3,5,6,7
2: 0,4,5,6,7
3: 1,5,6,7
4: 0,2,5,6,7
5: 0,1,2,3,4,6,7
6: 0,1,2,3,4,5,7
7: 0,1,2,3,4,5,6

(1,3 is an overused edge being in flat faces 1,3,7, 1,3,5, and 1,3,6)



0: 2,3,4,5,6,7
1: 2,3,4,5,6,7
2: 0,1,4,5,6,7
3: 0,1,4,5,6,7
4: 0,1,2,3,6,7
5: 0,1,2,3,7
6: 0,1,2,3,4,7
7: 0,1,2,3,4,5,6

(flat faces 1,3,5, 1,2,5, 0,3,5, and 0,2,5 create a dwarf rim about 5)

6.5.3 Obstructions on 10 vertices

As shown in the table of Figure 6.7 we find just 656 graphs on 10 vertices which are topological obstructions. Of these, 455 are minor order obstructions. The minor order obstruction on ten vertices with the most (30) edges (which happens also to be the topological obstruction on 10 vertices with the most edges) is presented in adjacency list form and is sketched in Figure 6.8. This interesting graph is highly symmetric, regular of degree 6, the line graph of K_5 (which itself is the obstruction to planarity with the most edges), and is the complement of the Petersen Graph. A 10 vertex obstruction must have at least 15 edges because the minimum vertex degree is 3; and no more than 31 edges because every single edge deletion from an 10 vertex graph with more than 31 vertices must be nontoroidal by excess edge ratio. There are about 15 million isomorphically distinct connected 10 vertex graphs on 11 to 31 edges. Processing these graphs takes several thousand hours (about 4000) on a mixture of Sun4 processors. All computing was done on Sun4 architectures, and there is a range of processor speeds. In order to compress the time taken to a few weeks, the problem was spread out over about thirty research machines and lab workstations. Because of the coarse grainedness of the testing project, this division of work did not present theoretical problems, though the pragmatic complexity of such a scheme is easy to underestimate. Essentially, each participating processor was given a segment of the interval of graphs to generate and test, and a "watchdog" program (in perl) implemented to monitor the use of the machines, their consoles, and the test and generate processes, to modify machine usage depending on type and density of traffic. (Lab workstations must be fully dedicated to Console users, but can be shared, at modified priority, when all users are remote.)

6.5.4 Obstructions on 11 vertices

Only a segment of the 11 vertex graphs were processed in this sub-project. There are roughly a billion isomorphically distinct 11 vertex graphs, and processing time increases sharply with the number of edges. Therefore the segment analyzed consists of the prefix segment of all 11 vertex graphs with 17 to 24 edges were processed. Obstructions could exist (and certainly do, witness the table 6.9) in the range of 25 to 34 edges. Processing these graphs takes several thousand hours (about 5000) on a mixture of Sun4 processors.

6.5.5 Obstructions with more than 11 vertices

Cursory searches for obstructions with more than 11 vertices were made. A twenty hour run on graphs with 12 vertices found 37 minor order obstructions, and the same amount of time on graphs with 13 vertices found a single minor order obstruction. (Eight minor order obstructions on 13 vertices were found in all. There are more.) Several hundred hours revealed no 14 vertex obstructions. Over 2,000 minor order obstructions have been found. A rough guess as to the total number of minor order obstructions to toroidality, informed by current research, would be about 10,000, though much larger numbers are not ruled out.

6.6 Correctness

The idea of the obstruction finding project is to test the toroidality tester. Aside from the fact that all "hand checked" graphs are being assigned the correct genus and obstruction status, two points are made.

First, the lists of toroidal obstructions on eight, nine, and ten vertices computed by an early version (August '92) of the program have been confirmed by a sequence of later versions, *even though* each later version uncovered bugs in some of the oldest routines. Similarly for the lists of minor order obstructions as of September '92.

Second, enquiries about such lists generated by others have been fruitless (although a few examples from Decker's thesis, [Dec78], have been checked, and a list of cubic irreducible graphs of genus one is rumoured to exist).

Uncertainty about the computed results however, is inevitable, and perhaps the greatest value of the lists is precisely as a basis for comparison with lists produced by other algorithms, when they are forthcoming.

Figure 6.6: Obstructions to toroidality—9 vertices

# edges	19	20	21	22	23	24	25	26		
# topol. obstruc.s	2	4	2	9	17	6	2	5	TOTAL	47
# minor o. obstruc.s	2	4	2	9	13	6	2	4	TOTAL	42

Figure 6.7: Obstructions to toroidality—10 vertices

# edges	19	20	21	22	23	24	25	26	27	28	29	30		
# top. obs.	14	8	34	40	190	170	102	75	21	1	0	1	TOTAL	656
# m. o. obs.	14	2	18	31	117	90	92	72	17	1	0	1	TOTAL	455

Figure 6.8: The ten vertex obstruction with the most edges

0: 2,3,4,6,7,8
1: 3,4,5,6,8,9
2: 0,4,5,7,8,9
3: 0,1,5,6,7,8
4: 0,1,2,6,8,9
5: 1,2,3,7,8,9
6: 0,1,3,4,7,9
7: 0,2,3,5,6,9
8: 0,1,2,3,4,5
9: 1,2,4,5,6,7

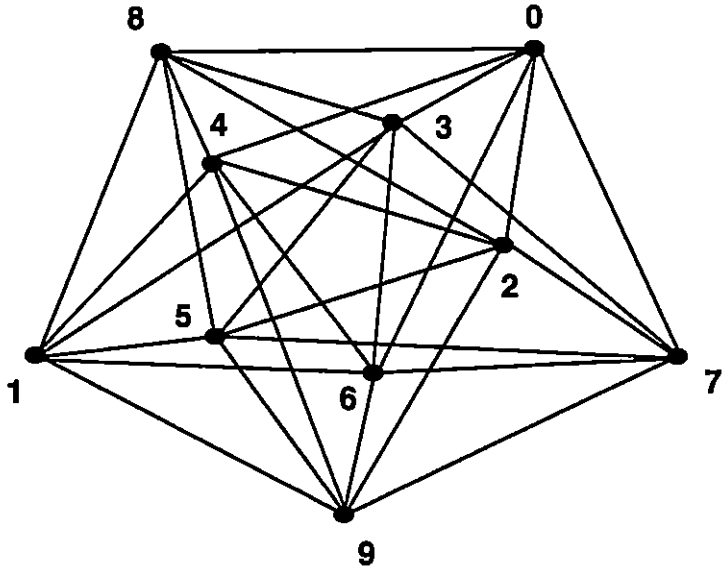


Figure 6.9: Some obstructions to toroidality—11 vertices

# edges	17	18	19	20	21	22	23	24	...		
# top. obs.	0	4	1	47	87	269	892	1878	...	(SUB)TOTAL	3178
# m. o. obs.	0	4	1	0	46	131	569	998	...	(SUB)TOTAL	1749

Chapter 7

Conclusions, Possible Future Developments

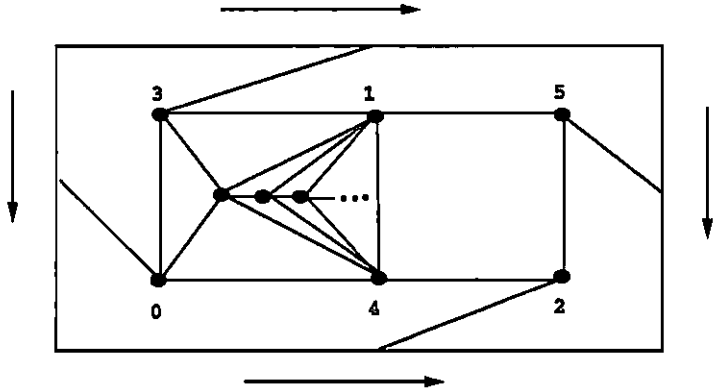
7.1 Status of the best current implementation

The current implementation of `e_toroidal` described in Chapter 5 is of exponential time complexity because of the need to test every bipartiton of the edges of attachment of every cycle tested. Performance of this algorithm deteriorates with increasing number of edges for a given number of vertices as well as, of course, increasing number of vertices. It is not clear how bad this deterioration for increasing number of vertices is in view of the applicability of the composition theorem for reduction.

It might be hoped that every large toroidal graph could be reduced to a graph of constant bounded size. If this were possible then an algorithm like $\mathcal{E}$ using such reductions could be analyzed to have sub-exponential time complexity, depending of course on the complexity of the reduction algorithm used. But this hope is not assured of realization in view of the existence of families of graphs such as that illustrated

in Figure 7.1. This embedding is of $K_{3,3}$ with an arbitrarily large "complication" in one 4-face of $K_{3,3}$. The graph is a toroidal block, has no "fat edges", and is not reducible by any application of the composition theorem, because it has no cycle whose removal leaves a nonplanar component. Therefore edge partitioning will occur with graphs with arbitrarily many vertices (and edges). Note that this graph is not a counterexample to the improvability of $\mathcal{E}$, since the first cycle selected in the minimum weight cycle basis may well be one with a bounded number of edges (for this family of graphs). Also, it is possible that more powerful reduction techniques may be found.

Figure 7.1: A family of graphs that do not reduce



7.1.1 Easy improvements to $\mathcal{E}$ as currently implemented

There are several ways to substantially improve the current implementation of algorithm $\mathcal{E}$, short of a modification which would result in a provably theoretically efficient tester.

- Currently `ftor` does not exploit the situation of having a list of $n_e - n_v$ flat faces at hand (but only exploits the flat face list to show nontoroidality when there are too many faces.) A simple routine to convert a set of faces to an

embedding and then check the genus would prove toroidality or nontoroidality here. This change would eliminate the need to process one (and in some cases more than one) more cycle with all its attendant bipartitions, and so be a definite improvement.

- Currently all reductions are cycle based. A preprocessor for the preprocessor could reduce fat edges before any cycle families are generated. This is a probable improvement.

- Triangulation as currently performed in `rpt` is simply based on using a single vertex of the face to be triangulated. Edges of triangulation already in the graph are not inserted and so triangulation does not always proceed as far as theoretically possible. An improved triangulation routine would try different pivot vertices when an edge is found to be already in the graph. This is a possible improvement. (Since `rpt` is recursive, "missed edges" might always be inserted later.) (Even though only chordless cycles are triangulated, many of them are triangulated at once.)

- Routine `rpt_tor` as implemented uses recursion (rather than deleting previously inserted edges) to backtrack. The method described in the thesis is a practical improvement.

- The family of cycles built for `ftor` need not "start from scratch" but can include the list of cycles which are flat faces found by `rpt`. This would result in a smaller family of cycles remaining for `ftor` to process.

- It may be possible to find more flat faces by randomizing the breadth first search producing the candidate cycles. Randomize **BFS** in the sense of randomizing the order of the list of vertices adjacent to the vertex being processed. Since

ordering the vertices for **BFS** by increasing degree results in more flat faces, there is reason to expect this method will yield even more. The question of how most efficiently to find all or most flat faces of a graph, is open.

- Routine `rpt` currently recurses whenever edges are added to the input graph. But since this recursion never seems to occur to a greater depth than two, an arbitrary decision not to recurse past a depth of one (or two) would save some unnecessary passes through `rpt`. If it turned out that extra edges are added for some graph in a third recursive call, such a graph would simply be passed to `ftor` with fewer flat faces found.

7.1.2 Improving on $\mathcal{E}$

More effective ways to densify a graph could be found in order to increase the likelihood that $G - C$ is 3-connected (and so having a single planar embedding, if any), thus allowing a faster determination of the contractibility status of C. This latter could be accomplished using the (unpublished) "double annulus" routines developed by the author in the Spring of '92. Note this improvement has not been fully implemented, has not been proven theoretically efficient, and graphs G which are not 3-connected are known to constitute a special case of inapplicability, for which special case however, specialized routines hold hope (ref. D.W. Barnette?).

7.1.3 Generalizations

Some of the results and algorithms of this thesis generalize readily for higher genus. The concept of contractibility, reviewed in Chapter 2, is already general. But the concept of flatness as defined and developed here is relative to the torus. In the

general case we would call *toroidal flat* what here has been called flat, and *genus k flat* a cycle which is contractible in every genus k embedding of the graph. Similarly for *genus k flat face*. The completion envisioned in the preceding subsection however is not readily generalizable to higher genus since 3-connectedness of a genus k graph does not imply unique genus k embeddability when $k > 0$. (Not even 6-connectedness guarantees unique embeddability in the torus: K_7 is 6-connected and any cycle can be chosen to be contractible or noncontractible.)

7.1.4 Future Work and Conclusion

It would be interesting to compare the tabulated results with those from some substantially different algorithm to get a higher degree of confidence and/or detect errors. When the "double annulus" project has been completed its performance and output on the obstruction search project can be compared with those of algorithm $\mathcal{E}$ as currently implemented. The ideal result is a practical polynomial time algorithm for torus embedding. It is hoped that some of the insights in this thesis will help the progress in this direction.

Bibliography

[AH89] Dan Archdeacon and Phil Huneke. A Kuratowski theorem for nonorientable surfaces. *Journal of Combinatorial Theory, Series B*, 46:173–231, 1989.

[BHKY62] 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.

[BM76] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* North-Holland, 1976.

[Col87] Charles J. Colbourn. *The Combinatorics of Network Reliability.* Oxford University Press, 1987.

[Dec78] Richard William Decker. *The Genus of Certain Graphs.* PhD thesis, The Ohio State University, 1978.

[DMP64] G. Demoucron, Y. Malgrange, and R. Pertuiset. Graphes planaires. *Rev. Française Recherche Opérationnelle*, 8:33–47, 1964.

[DR91] Hristo Djidjev and John Reif. An efficient algorithm for the genus problem with explicit construction of forbidden subgraphs. *ACM STOC*, 337–347, 1991.

[Edm60] J. Edmonds. A combinatorial representation for polyhedral surfaces. *American Mathematical Society Notices*, 7:646, 1960.

[Fil80] I.S. Filotti. An algorithm for embedding cubic graphs in the torus. *Journal of Computer and System Sciences*, 2:255–276, 1980.

[FL88] Michael R. Fellows and Michael A. Langston. Fast reduction algorithms for combinatorial problems of VLSI design. In J.H. Reif, editor, *VLSI ALGORITHMS AND ARCHITECTURES*, 278–287. 3rd Aegean Workshop on Computing, AWOC 88, Springer-Verlag, 1988.

[FMR79] I.S. Filotti, G.L. Miller, and J. Reif. On determining the genus of a graph in $o(v^{O(g)})$ steps. *Proceedings 11th Annual ACM Symp. Theory of Computing*, 27–37, 1979.

[Gib85] Alan Gibbons. *Algorithmic graph theory.* Cambridge University Press, 1985.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[GR79] Jonathon L. Gross and Ronald H. Rosen. A linear time planarity algorithm for 2-complexes. *Journal of the ACM*, 26:611–617, 1979.

[GR81] Jonathon L. Gross and Ronald H. Rosen. A combinatorial characterization of planar 2-complexes. *Colloquium Mathematicum*, 44:241–247, 1981.

[GT87] Jonathan L. Gross and Thomas W. Tucker. *Topological Graph Theory.* John Wiley and Sons, New York, 1987.

[Hef91] Lothar Heffter. Uber das problem der nachbargebiete. *Math. Ann.*, 38:477–580, 1891.

[Hor87] J. D. Horton. A polynomial time algorithm to find the shortest cycle basis of a graph. *SIAM Journal on Computing*, 16:358–366, 1987.

[HR76] Frank Harary and Ronald H. Rosen. On the planarity of 2-complexes. *Colloquium Mathematicum*, 36:101–108, 1976.

[HT74] J. Hopcroft and R. Tarjan. Efficient planarity testing. *JACM*, 21:549–568, 1974.

[Kur30] C. Kuratowski. Sur le problèm des courbes gauches en topologie. *Fund. Math.*, 15:271–83, 1930.

[LC83] C. I. Liu and Y. Chow. Enumeration of connected spanning subgraphs of a planar graph. *Acta Mathematica Hungarica*, 41:27–36, 1983.

[McK90] Brendan McKay. Technical report TR-CS-90-02. Australian National University, 1990.

[Mil80] Gary Miller. Isomorphism testing for graphs of bounded genus. *ACM*, 225–235, 1980.

[RP] N. Robertson and P.D.Seymour. Graph minors xvi. Wagner's conjecture. to appear.

[RS84] N. Robertson and P. Seymour. Generalising Kuratowski's theorem. *Congress. Numer.*, 45:129–138, 1984.

[RS85] N. Robertson and P. Seymour. Graph minors - a survey. In *Surveys in Combinatorics*. Cambridge University Press, 1985.

[SW86] Dennis Stanton and Dennis White. *Constructive Combinatorics*. Springer-Verlag, New York, 1986.

[Tho89] Carsten Thomassen. The graph genus problem is NP-complete. *Journal of Algorithms*, 10:568–576, 1989.

[Tho90] Carsten Thomassen. Embeddings of graphs with no short noncontractible cycles. *Journal of Combinatorial Theory, Series B*, 48:155–177, 1990.

[Tut62] W.T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13:743–68, 1962.

[Wag37] K. Wagner. Über einer eigenschaft der ebener complexe. *Math. Ann.*, 14:570–590, 1937.

[Whi32] H. Whitney. Congruent graphs and the connectivity of graphs. *American J. of Math.*, 54:150–168, 1932.

[Whi33] H. Whitney. 2-isomorphic graphs. *American J. of Math.*, 55:245–254, 1933.

[Wil85] S. Gill Williamson. *Combinatorics for Computer Science.* Computer Science Press, Rockville, Md., 1985.

[You63] J.W.T. Youngs. Minimal embeddings and the genus of a graph. *Journal of Mathematics and Mechanics*, 12:303–315, 1963.

VITA

Surname: **Neufeld** Given Names: **Eugene Trivan**
Place of Birth: **Coaldale, Alberta, Canada** Date of Birth: **October, 27th 1942**

Educational Institutions Attended:

University of British Columbia, Vancouver	1959 to 1963
University of Victoria	1989 to 1993

Degrees Awarded:

BA	1963	University of British Columbia, Vancouver, British Columbia

Partial Copyright License

I hereby grant the right to lend my thesis (the title of which is shown below) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis:

Practical Toroidality Testing

Author: [redacted]
Eugene T. Neufeld
June 30, 1993