Rotation Systems and Cellular Imbeddings

Alex Waldrop[*]

August 11, 2011

Abstract

This paper is broken up into two seperate parts. The first of which serves as an introduction to general rotation systems and imbedding graphs in 2-manifolds. It also discusses the various properties of the distrubution of the genus of these surfaces as well as some potential avenues for further research. The second part demonstrates a program written in Matlab that takes a graph and finds the distribution of the cellular imbeddings. Also included is the code adapted from a previous paper.

Part I

1 Introduction to Rotation Systems

First we begin with several definitions of rotation systems when dealing with a finte graph G.

Definition 1.1 *A graph G(V,E) is composed both by a vertex set V and and edge set E such that each edge $\in E$ is associated with a pair of verticies (v_i, v_j) which denotes the endpoints of each edge and a 0 or 1 telling if the edge is untwisted and twisted respectfully.*

Following from this definition we can see that in some cases $v_i = v_j$ making the edge a **loop**, beginning and ending at the same vertex. Also it is possible for two distinct edges e_i and e_j to be associated with the same pair of vertices. In this case we call these **multiple edges**. A **simple graph** is a graph G(V,E) s.t. there are no loops or multiple edges.

Now for a simple graph we can denote a **rotation** at a vertex $v_i \in V$ which is an ordered list unique up to a cyclic permutation of all the edges associated with that vertex (i.e. all the edges with an endpoint at v_i). For a simple graph each of these edges has another distinct endpoint different from v_i, thus we can instead use the vertex form of a rotation and make the rotation the ordered list of the verticies connected to v_i through an edge. From this point on we will

[*]University of Washington REU 2011

use this vertex form of a rotation. Let the **degree** d_i of a vertex v_i to be the number of edges connected to the vertex. If all of the edges are of type 0 then the number of rotations at v_i is equal to $(d_i - 1)!$.

Definition 1.2 *A pure rotation system $\mathfrak{P}_G(V,C)$ of a graph G(V,E) is the combined ordered lists $c_i \in C$ as defined above for each vertex $v_i \in V$ where all $e_i \in E$ are untwisted.*

By this definition we find that the number of pure rotation systems

$$|\mathfrak{P}_G| = \prod_{v_i \in V} (d_i - 1)!$$

Definition 1.3 *A general rotation system $\mathfrak{R}_G(V, C)$ of a graph G(V,E) is the combined ordered lists $c_i \in C$ as defined above for each vertex $v_i \in V \exists$ atleast one $e_i \in E$ that is twisted.*

Since each edge could be either twisted or untwisted

$$|\mathfrak{R}_G| = 2^{|V|}|\mathfrak{P}_G|$$

Example 1.4 *If we have the complete graph K_4 as shown above we can choose a rotation system such that all of the verticies adjacent are in a clockwise order in the lists. Then the rotation system will look like:*

$$\begin{array}{c} 1 : 2^0 4^0 3^0 \\ 2 : 4^0 3^0 1^0 \\ 3 : 1^0 2^0 4^0 \\ 4 : 3^0 1^0 2^0 \end{array}$$

If the edge between 3 and 4 was twisted the rotation system would look like:

$$\begin{array}{c} 1 : 2^0 4^0 3^0 \\ 2 : 4^0 3^0 1^0 \\ 3 : 1^0 2^0 4^1 \\ 4 : 3^1 1^0 2^0 \end{array}$$

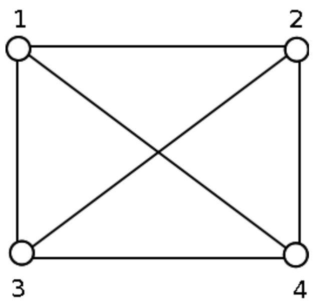


Figure 1: K_4 graph

A **projection of a rotation system** is a drawing of the graph G(V,E) such that all of the edges connected to each vertex are in a clockwise order based on the ordered list for each vertex. For example Figure 1 is the projection of the first rotation system by our construction of this system.

A projection with fixed verticies is unique for each rotation system. We will use this fact later on when discussing the overlap matrix.

2 Cellular Embedding

In an effort to reproduce the Cut Point Lemma from [1] for non-circular planar graphs we will now look at circular embeddings of graphs in 2-manifolds.

An **imbedding of a graph** G in an oriantable surface S is a continuos one-to-one function $\rho : G \rightarrow$ from a topological representation of the graph G into the surface S [4]. (In all imbeddings, the edges of the graph G do not intersect except at the verticies)

The imbedding is **cellular** if the interior of each face produced by the imbedding is homeomorphic to the 2-dimensional open disk. From this point on this will be our definition of imbedding.

There are two types of srfaces, orientable and non-orientable. The orientable surfaces are the sphere, torus and etc and are defined by their **genus**, in layman's terms, the number of holes.The higher genus orientable surfaces can be constructed by adding a *handle* to a surface to increase the genus by 1. The non-orientable surfaces that most people know are the mobius strip, the projective plane and the Klein bottle. These will be defined in terms of thier **crosscap number** similar to the genus of orientable sufaces.

Instead of thinking about these surfaces in 3-space we will now introduce a tool that will allow us to draw the imbeddings in the plane without edge crossing. The **fundamental polygons** are the polygons with paired directed edges that are pasted together to produce the genus n surface.

If we denote the direction of the edges in terms of a clockwise direction we can write the fundamental polygon in a succinct form. An orientable closed surface of genus n has the following standard fundamental polygon:

$$A_1 B_1 A_1^{-1} B_1^{-1} A_2 B_1 A_2^{-1} B_2^{-1} \cdots A_n B_1 A_n^{-1} B_n^{-1}$$

The torus is represented by $A_1 B_1 A_1^{-1} B_1^{-1}$ so orientable sufaces can be viewed as the glueing of tori together. This is analagous to pasting a handle to an existing surface.

A non-orientable surface of crosscap number n has the following standard fundamental polygon:

$$A_1 A_1 A_2 A_2 \cdots A_n A_n$$

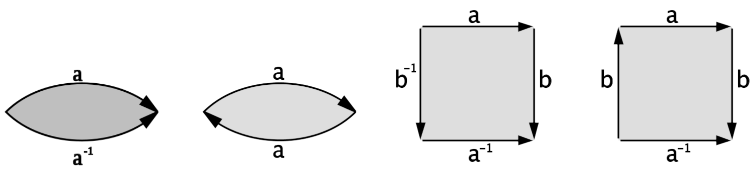
Alternately, the non-orientable surfaces can be given in one of two forms, as n Klein bottles glued together (this may be called the n-fold Klein bottle, with non-orientable genus 2n), or as n glued real projective planes (the n-fold crosscap, with non-orientable genus n). The n-fold Klein bottle is given by the 4n-sided polygon

$$A_1 B_1 A_1^{-1} B_1^{-1} A_2 B_1 A_2^{-1} B_2^{-1} \cdots A_n B_1 A_n^{-1} B_n$$

(note the final Bn is missing the superscript 1; this flip, as compared to the orientable case, being the source of the non-orientability). The 2n+1-fold crosscap is given by the 4n+2-sided polygon

$$A_1 B_1 A_1^{-1} B_1^{-1} A_2 B_1 A_2^{-1} B_2^{-1} \cdots A_n B_1 A_n^{-1} B_n^{-1} C^2$$

[5]



The 2-Sphere, Projective Plane, Torus, and Klein Bottle.

By combining the two concepts of rotation systems and cellular imbeddings we introduce a theorem.

Theorem 2.1 *Every pure rotation system for a graph G induces (up to a orientation-preserving equivalence of imbeddings) a unique imbedding of G into an oriented surface. Conversely, every imbedding of a graph G into an oriented surface induces a unique pure rotation system for G.*

Theorem 2.1 is ascribed to Heffter (1891) and Edmonds(1960).

Theorem 2.2 *Every rotation system on a graph G defines (up to equivalence of imbeddings) a unique locally orienred graph imbedding G→S. Conversely every locally oriented graph imbedding G→S defines a rotation system for G.*

Proof given in [2]

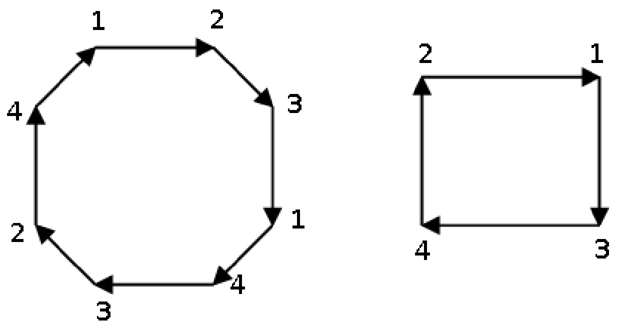
The basic question is that given a rotation system how do we imbed the graph into the surface S_n. The following algorithm will allow us to imbed graphs into both orientable and non-orientable surfaces (depending on the rotation system).

Face Tracing Algorithm *Assume that the given graph G has no 2-valent vertices. Choose an intial vertex v_0 of G and a first edge e_1 incident on v_0. Let v_1 be the other endpoint of e_1. The second edge e_2 in the boundary walk is the edge after (resp., before) e_1 at v_1 if e_1 is type 0 (resp., type 1). If the edge e_1 is a loop, then e_2 is the edge after (resp., before) the other occurence of e_1 at v_1. In general, if the walk traced so far ends the edge e_i at vertex v_i, then the next edge e_{i+1} is the edge after (resp., before) e_i at v_i if the walk so far is type*

0 (resp., type 1). The boundary walk is finished at edge e_n *if the next two edges in the walk would be* e_1 *and* e_2 *again. To start a different boundary walk, begin at the second edge of any corner that does not appear in any previously traced faces. If there are no unused corners, then all faces have been traced. By pasting the edges together in the created polygons we create a surface in which the graph is imbedded.*

Example 2.3 *Using the rotation system from 1.4 we will construct the boundary walks (polygons) that will be pasted together to form the surface.*

We begin at vertex 1 and choose to go to vertex 2 next. For now we have the boundary walk $1 \rightarrow 2$. *Then we look at the ordered list for vertex 2 and determine the next vertex as the one to the left of the previous vertex. Thus the next vertex is 3. Now our boundary walk is* $1 \rightarrow 2 \rightarrow 3$. *Continuing the process we get the boundary walk* $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 1$. *We notice that every edge has not been travelled twice (once each way) so we choose an edge that hasn't been travelled twice, for example the* $2 \rightarrow 1$ *edge and continue the algorith. By doing this we get another boundary walk,* $2 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 2$. *After this walk all edges have been traversed twice thus the algorithm has finished and produced two polygons, one octogon and one square as shown below. In this example, the genus of the surface S produced by the pasting of the polygons is 1, so the surface in which the complete graph* K_4 *for this particular rotation system is the torus.*



Remark: When dealing with rotation systems with twisted edges one must take care to switch from always going left to going right when crossing a twisted edge and switching back when crossing another twisted edge. To determine whether the rotation system produces a non-orientable imbedding, count the number of twisted edges in each polygon and if there is atleast one with an odd number of twisted edges then the surface is non-orientable.

To determine the genus of an embedding we introduce the "Euler characteristic of a cellular imbedding" of a connected graph as the value of the Euler formula $|V| - |E| + |F| = \chi(G \rightarrow S)$.

Theorem 2.4 *For each orientable surface* S_g *(g=0,1,2,...) there exists a connected graph G and a cellular imbedding* $G \rightarrow S_g$ *whose Euler characteristic satisfies the equation* $\chi(G \rightarrow S) = 2 - 2g$.

Theorem 2.5 *For each non-orientable surface N_k (k=0,1,2,...) there exists a connected graph G and a cellular imbedding $G \to N_k$ whose Euler characteristic satisfies the equation $\chi(G \to S) = 2 - k$.*

Proof: Proofs of these two theorems are found in [2]

Theorem 2.6 ***(The Invariance of Euler Characteristic of Orientable Surfaces)***
Let $G \to S_g$ be a cellular imbedding, for any g=0,1,2,... . Then $\chi(G \to S_g) = 2 - 2g$.

Proof: Proof by induction: First we must show that this is true for all imbeddings into the 2-sphere (g=0). We prove this by induction on the number of faces in the cellular imbedding.

If G is a tree then the number of faces equals 1 since there are no cycles and by the Jorder curve theorem the 2-sphere can not be split into more than 1 face homoemorphic to the unit disc. Thus for $|F_G| = 1, \chi(G \to S_0) = 2$ since for every tree $|V_G| - |E_G| = 1$. Now assume that the characteristic holds for a graph with $|F| = n$. Suppose that there is a graph G such that $|F_G| = n + 1$. Then there is some edge e that lies on the boundary between two distinct faces. Since they are distinct, the subgraph G' obtained by removing e is connected. Then $|F_{G'}| = |F_G| - 1 = n$. Also $|V_{G'}| = |V_G|$ and $|E_{G'}| = |E_G| - 1$, then by induction $|V_{G'}| - |E_{G'}| + |F_{G'}| = 2$. But the relations above show that $|V_G| - |E_G| + |F_G| = 2$. In conclusion the formula holds for the spher S_0.

As an inductive hypothesis, assume that it holds for the surface S_g , and suppose that a graph G is cellularly imbedded in S_{g+1}.

First draw a circle along one of the meridians of S_{g+1} such that it crosses the graph G at finitely many points and does not intersect any verticies and does not meet an edge more than once. Next thicken the circle to an annular region R. By excising the region R and capping of the two holes with discs we can obtain S_g. Now construct a cellular imbedding $G' \to S_g$. The vertex set of G' is the union of the vertex set of G and the intersection set of G with the two boundary circles of R. Suppose the graph G intersects the region R in p places then $|V_{G'}| = |V_G| + 2p$.

The edge set of G' contains all of the edges of G - p edges (the ones that intersect with R). Furthermore it contains the edges from the p intersection vertices to the graph outside of the region R so there are an additional 2p edges (a set for each side of R). In addition the edges that make up the circles that cap off the excised portion contribute another 2p edges (p for each circle). Thus $|E_{G'}| = |E_G| + 3p$.

Each face of the imbedding $G \to S_{g+1}$ is cellular. It follows from the Jordan curve theorem that excising the surface doubles the number of faces in between the region R and the rest of the graph G and that there were p faces in this region. Thus an additional p faces are created, along with the 2 circular faces that close of the open holes. Therefore, $|F_{G'}| = |F_G| + p + 2$.

Computing:

$$\chi(G \to S_g) = |V_{G'}| - |E_{G'}| + |F_{G'}|$$

$$= (|V_G| + 2p) - (|E_G| + 3p) + (|F_G| + p + 2)$$

$$= |V_G| - |E_G| + |F_G| + 2$$

$$= \chi(G \to S_{g+1}) + 2$$

By the induction hypothesis: $\chi(G' \to S_g) = 2 - 2g$
Implying that $\chi(G \to S_{g+1}) = -2g = 2 - 2(g+1)$
Proof done.

Theorem 2.7 ***The Invariance of Euler Characteristic of Non-Orientable Surfaces)***
Let $G \to N_k$ be a cellular imbedding, for any k = 0, 1, 2, ... Then $\chi(G \to N_k) = 2 - k$.

The proof of this theorem is very similar to proof of the previous proof. The only difference is instead of excising an annular region, one excises a mobius strip and does the same computations. For a full proof see Theorem 3.3.4 of [2].

3 Minimum and Maximum Genus

The study of the minimum genus γ(G) of the imbedding of a graph has been studied extensively for a long period of time, but recently there has been interest in the maximum genus γ_M(G) and the range of crosscap numbers $\bar{\gamma}$(G).

From the previous section's results, several facts follow immediately that allows us to draw some conclusions. First of which is that by Eulers formula for orientable surfaces $|V| - |E| + |F| = 2 - 2g$ that for a given Graph G, the genus g is maximized by the cellular imbedding $(G \to S_g)$ when $|F_G|$ is at its minimum. Likewise, the genus g is minimized by the cellular imbedding $(G \to S_g)$ when $|F_G|$ is at its maximum.

The number $|E| - |V| + 1$ is called the **cycle rank** (or the **Betti number** for a 2-manifold) of a graph G, denoted $\beta(G)$. Conceptually this is the number of edges in the cotree G - T of G (deleting the edges in tree T). $|F_G| \geq 1$ so by Eulers formula $\gamma(G) \leq \beta/2$.

Definition 3.1 *The **deficiency** ξ**(G)** of a graph G is the minimum number of odd components contained in the cotree G - T for each tree T of the graph G.*

Computation by exhaustion is not an efficient way to calculate $\xi(G)$ as the number of spanning trees grows exponentially as shown by the calculations using the Kirchoff matrix-tree theorem.

Theorem 3.1 ***Maximum Genus of a Graph***

$$\gamma_M(G) = \frac{1}{2}(\beta(G) - \xi(G)) \tag{1}$$

Proof found in [6]

Definition 3.2 *The* ***girth*** *of a graph G is the minimum length of a cycle in the graph.*

By our imbedding algorithm the minimum size of the polygon faces is equal to the girth of the graph and each edge is traced twice during the production of the polygons. Let $|F_n|$ be the number of faces with n sides. Then $2|E| = \sum_{i=1}^{n} i * |F_i|$. Combining this fact and that $i \geq girth(G)$ we see that $2|E| \geq girth(G) * |F|$ or $|F| \leq 2|E|/girth(G)$. If equal then $|F_G|$ is maximized, thus γ is mimimized. By pluggin in the inequality for $|F|$ in Euler's formula we get the equation:

$$\gamma(G) \geq \frac{(girth(G) - 2) * |E_G|}{2 * girth(G)} - \frac{|V_G|}{2} + 1 \tag{2}$$

Similarly with non-orientable embeddings we get the equation:

$$\bar{\gamma}(G) \geq 2 - |V_G| + frac(girth(G) - 2) * |E_G|girth(G) \tag{3}$$

Example 3.3 *Show that the complete graph K_5 is not planar.*

This is equivalent in saying that $\gamma(K_5) \geq 1$. For K_5, $|E| = 10$ and $|V| = 5$. The girth of any complete graph is 3 since every vertex is connected to every other vertex. Then by equation 2:

$$\gamma \geq \frac{(3-2) * 10}{2 * 3} - \frac{5}{2} + 1 = \frac{10}{6} - \frac{5}{2} + 1 = \frac{1}{6} > 0$$

Thus K_5 can not be planar, which means for all rotation systems of K_5 the genus γ of the imbedding $(K_5 \rightarrow S_g)$ is greater than 0.

Although this gives an idea of the value of the minimum genus of a graph, it generally does not give very useful input. For many graphs, using equation 2 results in $\gamma(G) \geq k$ where $k < 0$.

Theorem 3.4 ***(Additivity of Minimum Genus)***

Let $\{B_1, B_2, \cdots, B_k\}$ be the set of 2-connected *components of a graph G. Then*

$$\gamma_{min}(G) = \sum_{i=1}^{k} \gamma_{min}(B_i)$$

Proof. [7]

Theorem 3.5 ***(Additivity of Maximum Genus)***

Let $\{B_1, B_2, \cdots, B_k\}$ be the set of 2-edge-connected *components of a graph G. Then*

$$\gamma_{max}(G) = \sum_{i=1}^{k} \gamma_{max}(B_i)$$

Proof. [8]

Theorem 3.6 ***(Kuratowski's theorem)***

A graph is planar if and only if it contains no subgraph homeomorphic to either K_5 or $K_{3,3}$.

Proof. [9]

Theorem 3.7 .

A graph is planar if and only if it has neither K_5 nor $K_{3,3}$ as a minor.

Proof. [10]

Ideally there would be an analog for forbidden minors for the higher genus graphs, but even for the torus there are over a thousand forbidden minors. Not even all of the forbidden minors are known for the higher genera so this avenue of study has not been fruitful for determing the minimum genus.

4 Genus Distribution

Definition 4.1 *Two rotation systems $R_1(G)$ and $R_2(G)$ are call* ***adjacent systems*** *if and only if by the deletion of a single edge the new rotation systems (keeping all other things the same) $R_1^*(G')$ and $R_2^*(G')$ are equivalent.*

Remark: Two rotation systems are equivalent if the imbeddings are equivalent up to an orientation-preserving equivalence of imbedding.

Theorem 4.2 *Let $(G \to S)$ be a cellular graph imbedding, and let e be an edge of the graph G. Let F be the set of faces for $(G \to S)$, and let F' be the set of faces of the imbedding obtained by edge-deletion surgery at e. There are three cases:*

1. *The edge e is between two distinct faces in the imbedding. Then $|F'| = |F| - 1$*
2. *The face f is pasted to itself along edge e without a twist. Then $|F'| = |F| + 1$*
3. *The face f is pasted to itself along edge e with a twist. Then $|F'| = |F|$*

This theorem implies that the numbers of the faces in imbeddings of adjacent rotation systems both differ by at most one from the common number of faces in the imbeddings of G - e. Thus the imbeddings of R_1 and R_2 must differ by at most 2. This also implies an absence of gaps in the genus range.

In a similar manner there is an absence of gaps in the crosscap range. For the argument see [2].

From this absence of gaps we can draw the following two conclusions:

$\exists$ rotation system R(G) such that $\gamma(R(G)) = g$, for g such that

$$\gamma_{min}(G) \leq g \leq \gamma_M(G)$$

and

$\exists$ rotation system R(G) such that $\bar{\gamma}(R(G)) = k$, for k such that

$$\bar{\gamma}_{min}(G) \leq k \leq \bar{\gamma}_M(G)$$

Since we know that for a graph there exists rotation systems that produce imbeddings for each genus between the minimum genus up to the maximum genus, we may ask the question of how this are distributed over the genus (crosscap) range. There are several approaches to this problem. The first of which is to calculate the genus of all rotation systems of the graph and find the distribution explicitly. Another tool that was constructed is called the **overlap matrix** by B. Mohar [11].

Definition 4.3 ***Overlap Matrix M***

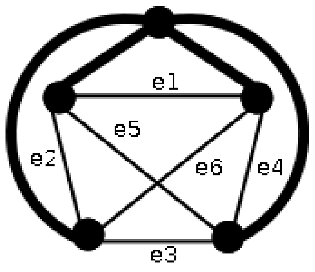
Let T be a spanning tree of a graph G, with edge sets E_T and E_G, respectively, and let $\{e_1, e_2, ..., e_p\}$ be the complete set of cotree edges, where β is the cycle rank of G. Let R be a general rotation system for G in which all edges in E_T are untwisted. The overlap matrix of R with respect to the spanning tree T is the $\beta x \beta$ matrix $M = [m_{i,j}]$ over GF(2) such that $m_{i,j} = 1$ if and only if either $i \neq j$ and the restriction of the underlying Pure rotation system to $T + e_i + e_j$ is nonplanar, or $i = j$ and e_i is twisted. If you draw a rotation projection in the plane for R so that the spanning tree T has no edge-crossings, then two different edges e_i and e_j overlap if and only if either they cross each other or one or both of them cross an edge of T. A twisted edge is defined to overlap itself.

[12]

Theorem 4.4 *Let R be a general rotation system for a graph, and let M be the overlap matrix. Then the rank of M equals twice the genus, if the corresponding imbedding surface is orientable, and it equals the crosscap number otherwise. It is independent of the choice of a spanning tree.*

Proof in [11].

Example 4.5 *We start with the projection of the rotation system of the complete graph K_5 and the tree T (shown in bold) and label the edges in the cotree as shown below.*



Following by the definition we can find the overlap matrix M(G, R, T). Notice that the tree T does not cross any edges so the only non-zero entries are when the edges in the cotree cross. From our projection we see that this only occurs when i=5 and j=6 or i=6 and j=5. The complete matrix M is:

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$Rank(M) = 2$ and since there are no twisted edges in this graph then the imbedding must be into an orientable surface. Thus by Theorem 4.4 the genus of this rotation system is 1 (i.e. R:$(G \to S_1)$).

Remark: All known distributions of graphs are unimodal. A sequence $\{a_m\}$ is **unimodal** if there exists and integer P such that

$$a_{m-1} \leq a_m \quad \forall m \leq P \quad \wedge \quad a_m \geq a_{m+1} \quad \forall m \geq P.$$

or equivalently

$$a_m^2 \geq a_{m+1}a_{m-1} \;\; \forall m$$

So if the sequence $\{g_m\}$ represents the number of rotation systems that produce an imbedding in genus m, $\{g_m\}$ is unimodal. This is an open problem as it is not known to be true in the general case of all graphs.

5 Miscellaneous and Research Ideas

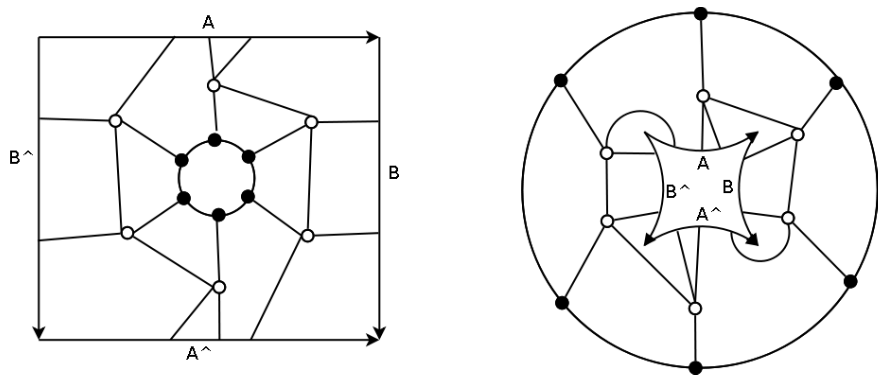
In this section I will be discussing some ideas that I have not been able to fully flesh out and could be potential avenues in research for anybody working in the REU or otherwise.

One area of thought was the effect of the Star-K transformation on the distribution of the genus. Since this technique is used in attempting to solve the inverse problem, I was looking for a correllation between the two distributions.

FACT: There exists a graph G such that a Star-K transformation changes the minimum (maximum) genus of the cellular embedding. By looking at equation 1, specifically the cycle rank, one can construct graphs that only change the deficiency by 1 and the cycle rank by a significant amount (ie more than 2).

One issue is that often the Star-K transformation produces graphs with multiple edges. If we merge the multiple edges we decrease the maximum genus. This was slightly dissapointing but I believe there might be more to this.

Another very interesting topic is generalizing the Cut Point Lemma from [1] to higher genus. Two papers that would be good reference are [13] and [14]. I will not go into detail here on the discussion on medial graphs and circular planar graphs as they are covered very explicitly in [1]. However, one thing that is important is the for electrical network graphs there are boundary nodes and interior nodes. The embeddings that we want to discuss should be such that all of the boundary nodes can be fit along a 2-cell homeomorphic to a unit disc with no edges in the interior of the disc. To acheive this using rotation systems, we add a vertex v to the vertex set of G and add edges $e_1, e_2, \ldots, e_n$ that have endpoints from the new vertex to the n boundary nodes. Then all imbeddings resulting from the rotation systems of G^* will be circular imbeddings in a 2-manifold. [14] gives examples of where the cut point lemma fails for annular graphs, but instead of looking at them in this way, I have considered putting the boundary verticies on the boundary of the disc and having the fundamental polygon inverted by a 1/z transformation as shown below. This might provide further insight and it bears a similar look to the circular planar medial graphs.



The annular graph G(3,2): torus polygon and 1/z transformation version.

This might be the most interesting and also difficult problem to tackle. To develop a similar lemma, even just for the case of graphs imbedded in the torus, would be great progess. This would give insight into the recoverablity of non-planar graphs, which is a very open question.

The last thing that I will discuss is the development of code to calculate the imbeddings of rotation systems. The second part of this paper will discuss my adaptation and improvement of Matlab code from [15]. Simply, this code takes a graph (in Adjacency matrix form) and produces all pure rotation systems and does the face trace algorithm for each one. In this way it produces the distribution of the graph across its imbeddable genera.

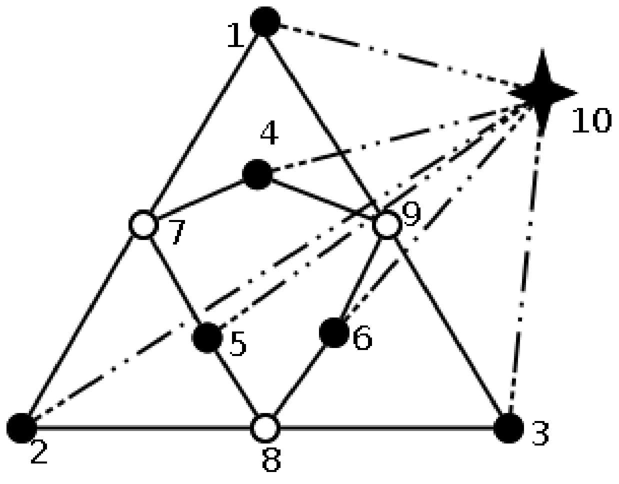
A potential improvement would be to adapt the program to do the same for non-orientable surfaces. That is, to include the possibility of twisted edges. This would increase the run time for even simple graphs by a factor of $2^{|V|}$ but it would still be an interesting program. The hardest thing to adapt would be the face trace algorithm to be able to switch from left to right when going over a twisted edge and how to represent the twisted edges as a number. Also writing the code in a more formidable language would be an improvement.

Part II

In this part I will discuss the improvements of the code found in [15] and present an example of the code.

The code that I have adapted can be found in the Appendix. I will not go into detail in the process of the code since it is covered extensively in [15]. However I will take the opportunity to show an example of the additions that I have made.

First consider the Triangle in Triangle graph as shown below. To produce a circular imbedding it requires the extra vertex (denoted by the star) and the extra edges from the star to the boundary verticies.



I want to produce an imbedding that preserves the order of the boundary verticies in a circular order around the new vertex. This is equivalent to the rotation around the new vertex to be [1 2 3 4 5 6]. [15] discusses this problem but does not provide the code to actually accomplish this. I have rewritten the program to calculate all rotation systems from the array of all rotations for each vertex (as a matrix). The function crunch.m produces the array of rotations for each vertex, but I have added a prompt for the user to change the rotations at any number of verticies. Not only does this only produce imbeddings that we desire, it reduces the number of rotation matricies that are computed. The number of pure rotation systems for the Triangle in Triangle graph without any restrictions is equal to 1,658,880 and takes about 12 minutes to compute all of the imbeddings. If we restrict the rotations as described above we reduce the number of systems to 13,824.

Example 5.1 *Taking the above graph in a circular order as described, we will run it through the program and find the imbedding with the minimum genus and produce the polygons that would be pasted together.*

The adjacency matrix for the triangle in triangle graph is:

$$K = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Now we run the genus2.m file.

```
genus2(K,'TTcirc')
Do you want to edit? Y/N
Enter Vector and Array {[10], {[1 2 3 4 5 6]}}
Total Distinct Rotation Systems:
       13824

Maximum Genus:
     4

Minimum Genus:
     2

Possible Ways to Embed in That Genus:
   249
```

This produces a table where the first column is the rotation matrix number, the second column is the number of rotation matricies that that have the same *polygon distribution. The rest are the number of edges in the polygon faces, where zero implies there isnt any more faces.*

facetable=

1	1794	30	6	0	0	0	0
2	954	28	8	0	0	0	0
3	192	12	10	8	6	0	0
4	1095	24	12	0	0	0	0
5	795	22	14	0	0	0	0
7	828	26	10	0	0	0	0
8	801	20	16	0	0	0	0
17	366	22	6	4	4	0	0
18	348	20	8	4	4	0	0
19	240	18	8	6	4	0	0
20	2418	32	4	0	0	0	0
21	132	14	14	4	4	0	0
31	363	18	18	0	0	0	0
34	18	12	8	8	8	0	0
49	414	24	4	4	4	0	0
50	15	8	8	8	4	4	4
201	288	16	10	6	4	0	0
203	303	20	6	6	4	0	0
205	204	12	10	10	4	0	0
209	300	14	12	6	4	0	0
210	183	12	12	8	4	0	0
221	348	16	12	4	4	0	0
250	402	18	10	4	4	0	0
385	204	16	8	6	6	0	0
386	129	14	8	8	6	0	0
800	111	14	10	6	6	0	0
818	57	16	8	8	4	0	0
820	108	14	10	8	4	0	0
961	81	18	6	6	6	0	0
1539	18	10	6	6	6	4	4
1547	9	12	6	6	4	4	4
1747	6	8	8	6	6	4	4
2116	90	12	12	6	6	0	0
2695	39	14	6	4	4	4	4
2696	60	10	8	6	4	4	4
2703	27	16	4	4	4	4	4
2719	15	10	10	4	4	4	4
2903	54	12	8	4	4	4	4
3683	9	10	10	10	6	0	0
11386	6	8	6	6	6	6	4

```
 >> load TTcirc
>> readrot11386=read_rot(11386, 10, rotmatrix)
```

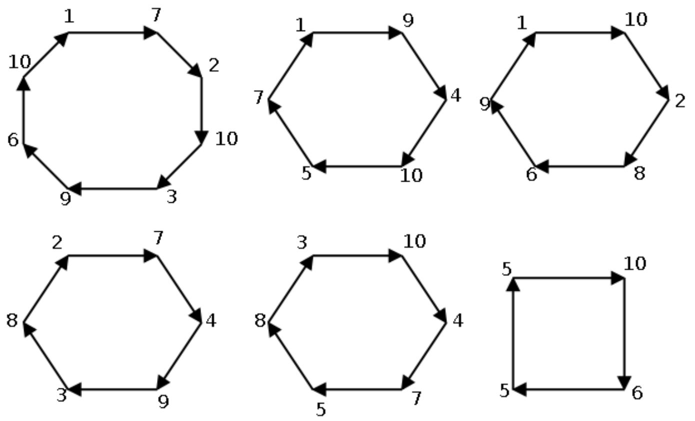
```
readrot11386 =
     7     9    10     0     0     0
     7    10     8     0     0     0
     8    10     9     0     0     0
     7     9    10     0     0     0
     7     8    10     0     0     0
     8     9    10     0     0     0
     1     2     4     5     0     0
     2     6     5     3     0     0
     1     4     3     6     0     0
     1     2     3     4     5     6
>> makefaces(readrot11386)
ans =
     1     7     2    10     3     9     6    10
     1     9     4    10     5     7     0     0
     1    10     2     8     6     9     0     0
     2     7     4     9     3     8     0     0
     3    10     4     7     5     8     0     0
     5    10     6     8     0     0     0     0

>> time
time =
   30.2286
```

Readrot.m produces the rotation matrix. For this example I choose the one that has the minimum genus of 2 (since it has the maximum number of faces).

The makefaces.m produces the polygons with each row being a polygon and the entries are the vertices.

We see that this only takes 30 seconds to compute the faces for all the rotation matricies. This is a significant decrease in run time in comparison, but we did not get an imbedding that is in the torus.

Below are the 6 polygons produced by this rotation matrix.



Improving this code to include twisted edges would be a significant acheivement but might require an overhaul of all of the code. However this is a very nice way to look at the genus distribution and even the polygonal distribution which is an even harder problem.

The Matlab files in the Appendix should be available through the UW Math REU webpage.

References

[1] E. B. Curtis and J. A. Morrow, Inverse Problems for Electrical Networks, Series on Applied Mathematics Vol. 13, World Scienctific Publishing (2000).

[2] J. L. Gross and T. W. Tucker, Topological Graph Theory, New York Dover (2001).

[3] A. T. White, Graphs, Groups and Surfaces, North-Holland Math. Studies Volume 8 (1972)

[4] J. L. Gross, Topics in Graph Theory (COMS 6204) Lecture Notes, Spring 2010

[5] Alan F. Beardon, The Geometry of Discrete Groups (1983), Springer-Verlag, New York

[6] N.H. Xuong , How to determine the maximum genus of a graph. J. Combin. Theory Ser. B 26 (1979), pp. 216225.

[7] J. Battle, F. Harary, Y. Kodama, J. W. T. Youngs, Additivity of the genus of a graph, Bull. Amer. Math. Soc. 68 (1962), 565-568

[8] E. Nordhaus, B. Stewart, and A. White, On the maximum genus of a graph, J. Combin. Theory Ser. B 11 (1971), 258-267

[9] K. Kuratowski, Sur le probleme des courbesgauches en topologie, Fund. Math. 15 (1930), 271-283

[10] K. Wagner, Uber eine Eigenschaft der ebenen Komplexe, Math. Ann. 114 (1937), 570-590

[11] B. Mohar, An obstruction to embedding graphs in surfaces, Discrete Math. 78 (1989) 135-142

[12] J. Chen et al, Overlap matrices and total imbedding distributions, Discrete Math. 128 (1994) 73-94

[13] N. Reichert, "Generalized Circular Medial Graphs", UW Math REU 2004

[14] Z. Geballe, "How Extra Connections Ruin the Cut Point Lemma", UW Math REU 2006

[15] O. Biesel and J. Eaton, "Notes on Multiple Embeddings", UW Math REU 2005

6 Appendix

A1 genus2.m

```
function [facetable, rotmatrix] = genus2(K, filename)
% GENUS2(K) Calculates the minimal genus of a graph with adjacency matrix
%
% K is a symmetric adjaceny matrix with K(i,j) = 1 if there is an edge
% between nodes i and j and K(i,j) = 0 otherwise. If K is not a valid
% adjacency matrix, CRUNCH(rot) will throw an error from faulty
% calculation.
%
% GENUS2(K) outputs the minimum genus of an embedding of a graph with
% adjacency matrix K possible embedding, as well as the total number of
% possible embeddings and the total number of possible embeddings in the
% minimal genus.
% Sets a timer to see how long the it takes to run the program
tic;
maxvalence = 0;
vert = size(K,1);
horiz = size(K,2);
edges = 0;

% Calculates the largest degree of any node.
for i=1:vert
    counter = 0;
    for j=1: horiz
        if K(i,j) ~= 0
        counter = counter + 1;
        end
    end
    if counter > maxvalence
    maxvalence = counter;
    end
end

% Calculates the number of edges in K
for i = 1:vert
    for j = 1: horiz
        if K(i,j) ~= 0
            edges = edges + 1;
        end
    end
end
edges = edges/2;
```

```
% Constructs a rotation system L from K.
%Basically puts verticies in numerical order and zeroes if valence less
%then maxvalence.
L = zeros(vert, maxvalence);
for i = 1:vert
counter = 1;
    for j = 1:horiz
        if K(i,j) ~= 0
        L(i, counter) = j;
        counter = counter + 1;
        end
    end
end

% Calls allrots to construct an array of all distinct rotation systems of K.
Y=allrots(L);

%Creates a matrix all rotation systems of K as row entries
rotmatrix=zeros(size(Y,2), size(Y{1,1},1)*size(Y{1,1},2));
for i=1:size(Y,2);
%Uses tocol to convert matrix to column
rotmatrix(i,:)=tocol(Y{1,i})';
end

% Calls GENUS_FACE to find the genus of each rotation system and records the
% minimal genus and the number of embeddings in that genus.
distinct_rotations = size(rotmatrix,1);
mingenus = genus_face(L);
totalmin = 0;
maxfaces = (edges-vert+2);
allfaces = zeros(0, maxfaces);
face_occur = zeros(0,1);
face_index = zeros(0,1);
for i = 1:distinct_rotations
genface = genus_face(read_rot(i, vert, rotmatrix));
genus = genface(1);
face = genface(2:end);
    if genus == mingenus
    totalmin = totalmin + 1;
    end
    if genus < mingenus
    mingenus = genus;
    totalmin = 1;
    end
newface = true;
    for j = 1:size(allfaces,1)
```

```
        if face == allfaces(j, 1:size(face,2))
        newface = false;
        face_occur(j, 1) = face_occur(j, 1) + 1;
break;
        end
    end
    if newface == true
    allfaces = [allfaces; zeros(1, maxfaces)];
    allfaces(end, 1:size(face, 2)) = face;
    face_occur = [face_occur; 1];
    face_index = [face_index; i];
    end
end

% Prints some useful information.
disp('Total Distinct Rotation Systems:')
disp(distinct_rotations);
disp('Minimum Genus:')
disp(mingenus);
disp('Possible Ways to Embed in That Genus:')
disp(totalmin);
% Constructs an array ?facetable? that encodes in the first column a row
% index corresponding to a rotation system, the second column is the number
% of times the given face system appears, and each remaining non-zero entry
% corresponds to the degree of one face in the embedding.
facetable = [face_index face_occur allfaces(:, 1:end - 2*mingenus)];
% Ends the time counter
time = toc;
% Saves the specified variables to the file specified.
save(filename, 'K', 'facetable', 'rotmatrix', 'time');
```

A2 genusface.m

```
function g = genus_face(rotsyst)

% Returns the genus and degree of the faces of the graph given rotsyst.
faces = 0;
vertices = size(rotsyst,1);
maxvalence = size(rotsyst,2);
edges = 0;

% Counts the number of edges in the graph.
for(i = 1: vertices)
    for(j = 1: maxvalence)
        if rotsyst(i,j) ~= 0
        edges = edges + 1;
        end
    end
end
edges = edges/2;

% Constructs a matrix the same size as the rotation system to track steps.
C = zeros(vertices,maxvalence);

% ?facelist? is a vector that stores the degree of each face as the algorithm
% uses rotation systems to trace them.
facelist = [];

% Uses algorithm described in Beisel and Eaton "Notes on Multiple
% Emebeddings" to trace each face created by the embedding.
for(i = 1: vertices)
    for(j = 1: maxvalence)
        if rotsyst(i, j) ~=0 && C(i, j) == 0
        prev = 0;
        faces = faces + 1;
        startvertex = i;
        C(i, j) = rotsyst(i, j);
        t = j - 1;
            if t == 0
            t = maxvalence;
            end
            while rotsyst(i, t) == 0
            t = t - 1;
            end
        prev = rotsyst(i, t);
        currentvertex = i;
        loopvertex = rotsyst(i,j);
        edgecounter = 1;
            while ~(loopvertex == startvertex && prev == currentvertex)
            edgecounter = edgecounter + 1;
            counter = 1;
                while rotsyst(loopvertex,counter) ~= currentvertex
                counter = counter + 1;
                end
            currentvertex = loopvertex;
            counter = mod(counter, maxvalence) + 1;
                while rotsyst(currentvertex, counter) == 0
                counter = mod(counter, maxvalence) + 1;
                end
            C(currentvertex, counter) = rotsyst(currentvertex, counter);
            loopvertex = rotsyst(currentvertex, counter);
            end
        facelist = [facelist, edgecounter];
        end
    end
end

% Sorts the facelist in descending order for comparison.
facelist = -sort(-facelist);
% Returns a vector giving the minimum genus as the first entry and the
% facelist as the rest of the vector.
g = [(2 - vertices + edges - faces)/2, facelist];
```

A3 readrot.m

```
function G = read_rot(i, num_vert, allperms)
% READ_ROT(i, allperms) converts a row of the allperms array into a
% rotation matrix.
max_degree = size(allperms, 2)/num_vert;
for j = 1:num_vert;
G(j,1:max_degree) = allperms(i, (max_degree*(j-1)+1):max_degree*j);
end
```

A4 makefaces

```
function facetrace = makefaces(rotsyst)
% Traces the vertice of each face of embedding given by ?rotsyst?.
%
% Almost identical to GENUS_FACE (refer to that for comments). Instead of
% counting mingenus and face degrees, each row of ?faces? tracks each
% vertex in one face of the embedding given by ?rotsyst?.
facetrace=[];
faces = 0;
vertices = size(rotsyst,1);
maxvalence = size(rotsyst,2);
edges = 0;
for i = 1:vertices
for j = 1:maxvalence
if rotsyst(i,j) ~= 0
edges = edges + 1;
end
end
end
edges = edges/2;
C = zeros(vertices,maxvalence);
row = 0;
column = 1;
for i = 1:vertices
for j = 1: maxvalence
if rotsyst(i, j) ~=0 && C(i, j) == 0
```

```
prev = 0;
faces = faces + 1;
startvertex = i;
C(i, j) = rotsyst(i, j);
t = j - 1;
if t == 0
t = maxvalence;
end
while rotsyst(i, t) == 0
t = t - 1;
end
prev = rotsyst(i, t);
currentvertex = i;
column = 1;
row = row + 1;
facetrace(row, column) = currentvertex;
loopvertex = rotsyst(i,j);
while ~(loopvertex == startvertex && prev == currentvertex)
counter = 1;
while rotsyst(loopvertex,counter) ~= currentvertex
counter = counter + 1;
end
currentvertex = loopvertex;
column = column + 1;
facetrace(row, column) = currentvertex;
counter = mod(counter, maxvalence) + 1;
while rotsyst(currentvertex, counter) == 0
counter = mod(counter, maxvalence) + 1;
end
C(currentvertex, counter) = rotsyst(currentvertex, counter);
loopvertex = rotsyst(currentvertex, counter);
end
end
end
end
```

A5 allrots.m

```
function Y = allrots(rotsys)
%Calls crunch to create all cyclic permutations of the rotations
%at each vertex.
A=crunch(rotsys);
%Creates cell array of all combinations of these rotations.
Y=cell(1,numrotsys(A));
for i=1:size(Y,2)
    b=1;
    for j=1:size(rotsys,1)
    b=b*size(A{1,j},1);
    h=size(A{1,j},1);
    Y{1,i}(j,:)=A{1,j}(mod(ceil((2*i-1)/b),h)+1,:);
    end
end
```

A6 numrotsys.m

```
function G = numrotsys(carray)
G=1;
for i=1:size(carray,2);
    G=G*size(carray{1,i},1);
end
```

A7 crunch.m

```
function [C, varargout] = crunch(rotsys)
%Creates array 1 by number of verticies
C = cell(1,nargout);
%Populates array with matrix where each row is a cyclic permutation
%of the rotation about that vertex
%Calls cyclic perms to do this
for i=1:size(rotsys,1);
    f = cyclicperms(rotsys(i, :));
    C{1,i} = f;
end
%This prompt allows the user to determine certain rotations
%The vector is the vertices and the array is the matrix of rotation
reply1=input('Do you want to edit? Y/N', 's');
if isempty(reply1);
    reply1='Y';
        reply2=input('Enter Vector and Array');
        a=reply2{1,1};
        b=reply2{1,2};
        for i=1:size(a,2);
        C(1,a(1,i))=b(1,i);
        end
end
```

A8 cyclicperms.m

```
function G = cyclicperms(row)
% Calculates all cyclical permuations of non-zero entries in row vector.
%
% CYCLICPERMS(row) reads a row vector of distinct non-zero positive
% integers followed by any number of zeroes and returns a matrix of all
% cyclic permuations of the non-zero integers (leaves zeroes at right of
% matrix).
% Creates the subset of the row vector to be permuted
if min(row) ~= 0
permutables = row(2:end);
else
permutables = row(2:min(find(row==0))-1);
end
num_permed = size(permutables,2);
% This reverses the order so the ?perms? function gives a nice looking
% output.
permutables = permutables(num_permed:-1:1);
% Creates a matrix of all cyclic permutations of the argument vector.
G(:,2:num_permed+1) = perms(permutables);
G(:,1)=row(1);
G(:,num_permed+2:size(row,2))=0;
```

A9 tocol.m

```
function X = tocol( X )
% TOCOL Converts a vector or a matrix into a column vector.
%   If input is already a column vector, it is returned with no change.
%   If input is a row vector, it is converted into a column vector and
%   returned.
%   If input is a matrix, each row is converted into a column, and all
%   resulting columns are placed in series into a single column which is
%   returned.
% Input:
%   X - input vector or matrix
% Output:
%   X - column vector
% check if input is a vector
    [ m, n ] = size(X);
    if m*n==m
        return % input is already a column vector with n rows
    elseif m*n==n
        X = X'; % input is converted from row vector to column vector
    elseif (m*n>n) || (m*n>m)
        X = X';
        X = X(:); % input is converted from matrix to column vector by row
    else
        X = []; % input is unknown and an empty output is returned
    end
end
```