Notes on Multiple Embeddings

Owen Biesel and Jeff Eaton

July 18, 2005

Contents

1 Producing Embeddings From Rotation Systems

We begin with a general discussion of graph embeddings (on orientable surfaces) and their production. By definition, an orientable surface makes sense of an outward-pointing normal vector, to which we can attach a global sense of clockwise and counterclockwise. We may therefore make sense out of a counterclockwise ordering of the neighboring vertices, for each node in the graph. This collection of cyclical orderings is called a *Rotation System*. Clearly, changing the particular embedding of the graph could change the rotation system. However, it is not immediately obvious that the rotation system alone completely determines the entire embedding of the graph, and even the genus of the host surface. This is accomplished by using the rotation system to explicitly write down the faces determined by the embedding. Consider the following rotation system for a graph with 8 vertices:

$$\begin{array}{lllll}
1: & 5 & 8 & 7 & \\
2: & 5 & 8 & 6 & \\
3: & 6 & 8 & 7 & \\
4: & 5 & 6 & 7 & 8 \\
5: & 1 & 2 & 4 & \\
6: & 2 & 3 & 4 & \\
7: & 1 & 4 & 3 & \\
8: & 1 & 2 & 3 & 4
\end{array}$$

Where the notation indicates that following the edges counterclockwise around vertex 5, for instance, one encounters edges leading to vertices 1, 2, and 4, in that (cyclical) order. We can now use this information to reconstruct the faces of the embedded graph. First, select any edge to begin with; we select the edge from vertex 1 to vertex 5. In our rotation system, at node 5, the vertex 1 is immediately followed by vertex 2 (this means that one corner of a face follows vertices 1 to 5 to 2). Therefore the next edge is the edge from node 5 to node 2. Similarly, nodes 8, 3, 7, 1, and 5 follow. But edge 1 to 5 is the edge we began on, so any further progression would merely yield the same edges over again. Therefore, the cycle

$$1 \to 5 \to 2 \to 8 \to 3 \to 7 \to \dots$$

gives a six-sided face in the embedding of the rotation system. Similarly, the other faces are

$$\begin{array}{l}
1 \to 8 \to 2 \to 6 \to 3 \to 8 \to 4 \to 5 \\
1 \to 7 \to 4 \to 8 \\
2 \to 5 \to 4 \to 6 \\
3 \to 6 \to 4 \to 7
\end{array}$$

Note that all the edges here have been used exactly twice, once in either direction. This indicates that each edge will always bound two different sides in the system of faces. Also note that the rotation system has determined exactly 5 faces. We may use this fact in Euler's formula to find the genus of the surface in which the graph is embedded:

$$\begin{array}{rcl}
2 - 2g & = & V - E + F \\
& = & 8 - 13 + 5 \\
& = & 0 \\
& \Rightarrow & g = 1
\end{array}$$

Therefore the graph has been embedded on a torus. (We will meet this graph again in Section 2, where the above embedding is discussed in more detail.)
The above process may be followed for any rotation system of a graph, and so all possible embedding of a graph in a surface of any arbitrary genus may be found. Therefore we may calculate the minimum such genus, called the genus of the graph. This process is employed in full in Section 3.

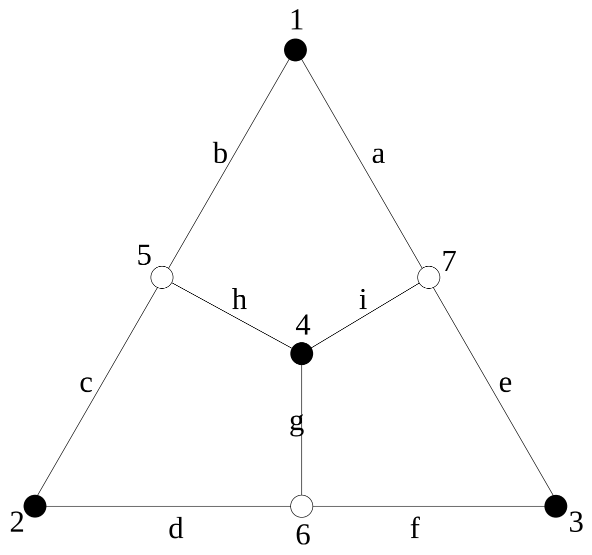


Figure 1: An annular planar representation of the 4-3-9 graph with vertices and edges labeled.

2 The 4-3-9 Graph

Figure 1 shows a graph which we have spent some time studying. This graph has the unusual property that, while not too large, it has interesting properties in its different embeddings. We call it the 4-3-9 graph because it has four boundary nodes, three interior nodes, and nine edges. The graph is annular planar (see [2] for more on annular planar graphs), because it may be embedded in the plane such that its boundary vertices lie on two concentric circles, and the interior of the graph lies completely inside the annulus formed by the circles. Note that the graph is clearly non-recoverable, but here we do not discuss the graph's electrical properties. Rather, we will analyze the graph from a topological perspective.

2.1 Genus 1 Embeddings

The 4-3-9 graph is cellularly embeddable in the torus with all of the boundary nodes lying on a circle. We call this a circular cellular embedding. In practice, embeddings of this type are obtained by affixing an extra node to the graph, joined by a single edge to each boundary node. (In the figures, this is denoted by a gray node with the number 8.) Any cellular embedding of the new graph will ensure that there exists a circle on which all the boundary nodes lie, and the interior of which contains only those edges joined to the extra node. Removing this catalyst yields a circular embedding. Note that the circular embedding is by no means unique; Figure 2 shows three distinct embeddings of the 4-3-9 graph on the torus, shown here in its usual square representation.

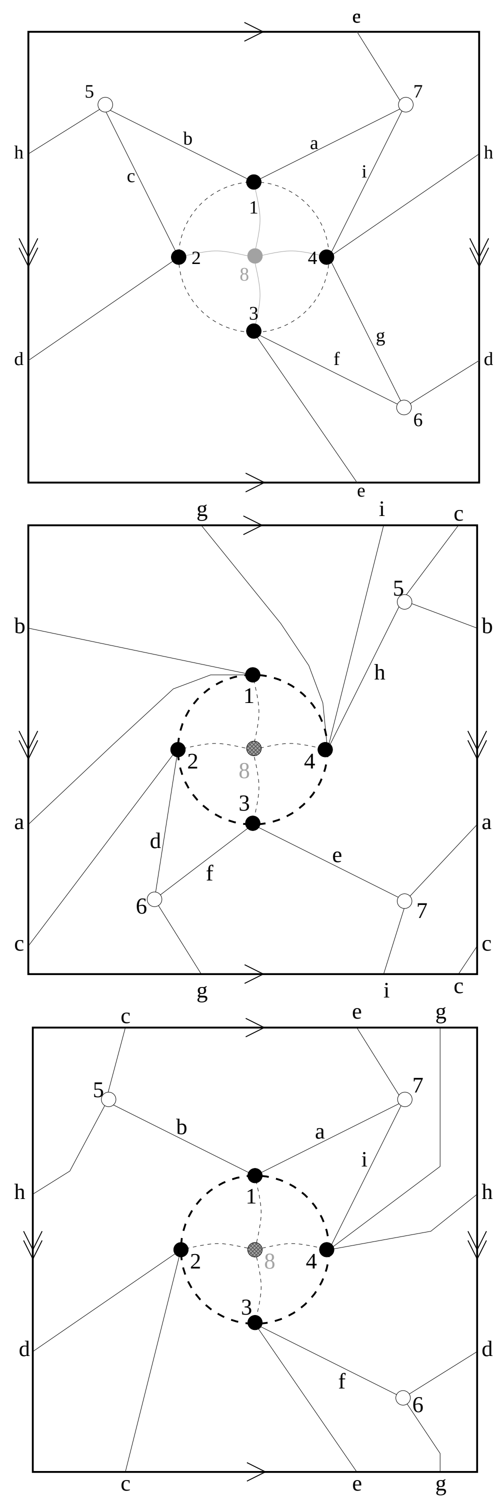


Figure 2: Three different embeddings of the 4-3-9 graph in Figure 1 into the torus. The lettering and numbering of the vertices and edges are the same as those of Figure 1.

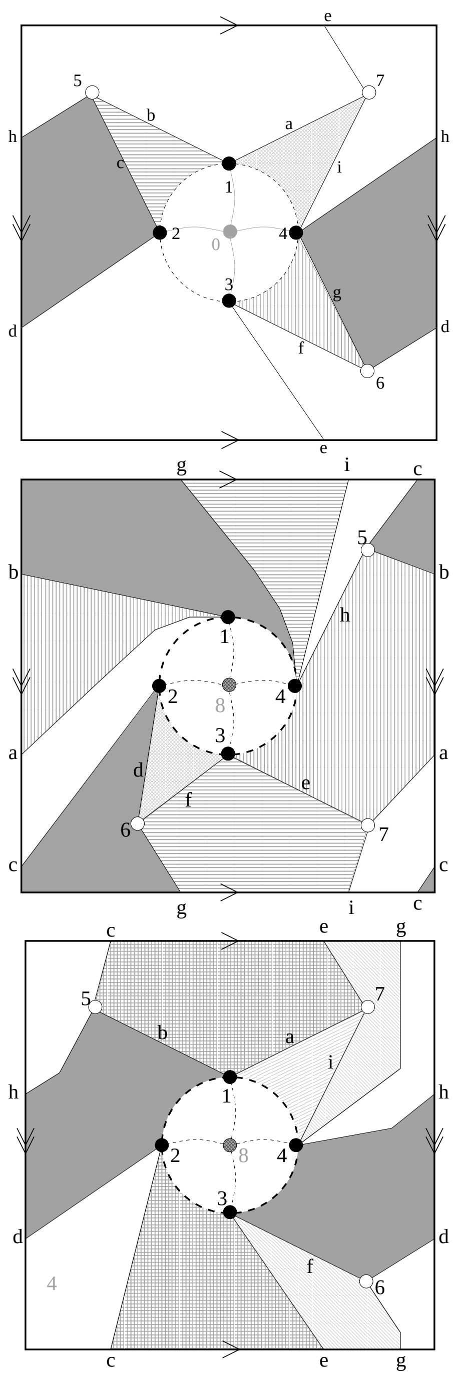


Figure 3: Shadings of the embeddings in Figure 2. Each shading represents a single face, and accordingly one vertex of the dual graph. We characterize such radically different embeddings by their faces; the leftmost graph is called a 10 4 4 4 4 embedding because it comprises one decagon and four quadrilaterals. Similarly, the center graph is a 6 6 6 4 4 embedding, and the rightmost graph is an 8 6 4 4 4 embedding. Embeddings of the same graph which share this numbering convention are termed *Quasi-Similar* embeddings.

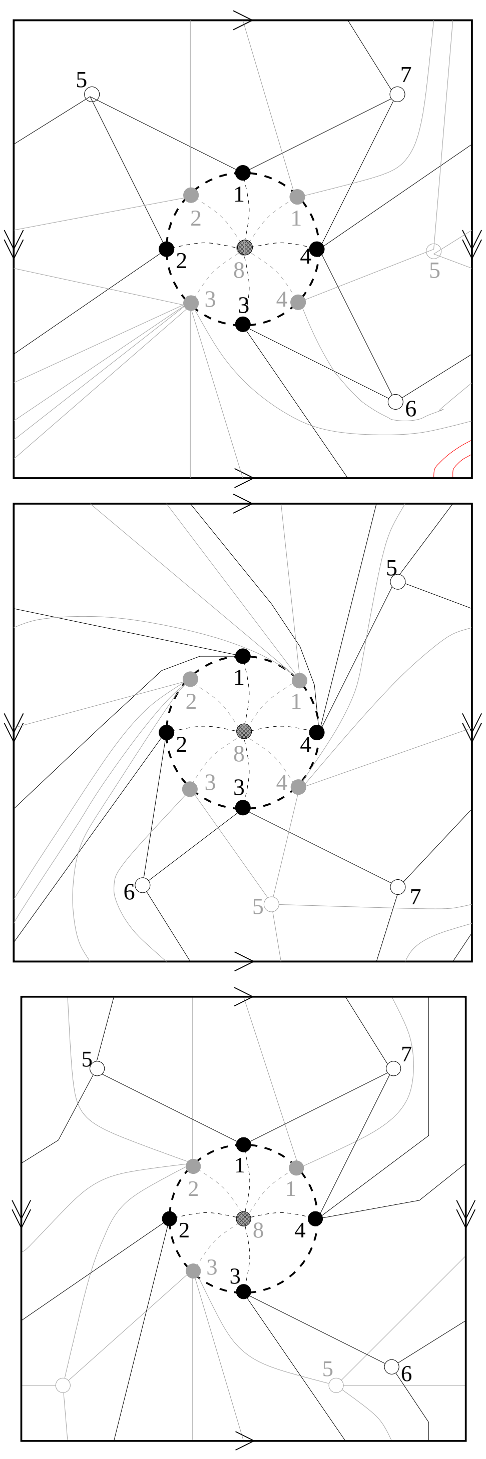


Figure 4: The embeddings of the 4-3-9 graph are shown again here, along with their respective duals, shown in gray.

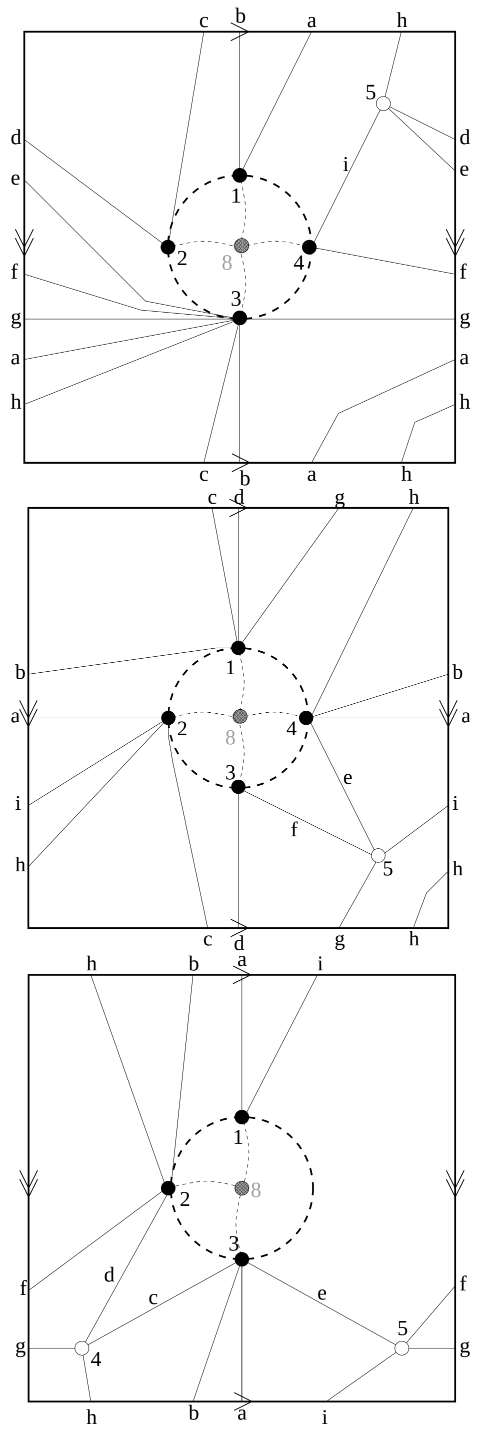


Figure 5: Here we have more readable drawings of the dual graphs shown in Figure 4. Note that they are fundamentally distinct, because their vertices have radically different degrees and connections. Also note that the final dual graph contains only three boundary nodes (but has an extra interior node). This is because of the way the octagon in the 8 6 4 4 4 wraps around the torus, and so can only contain a single vertex in the dual.

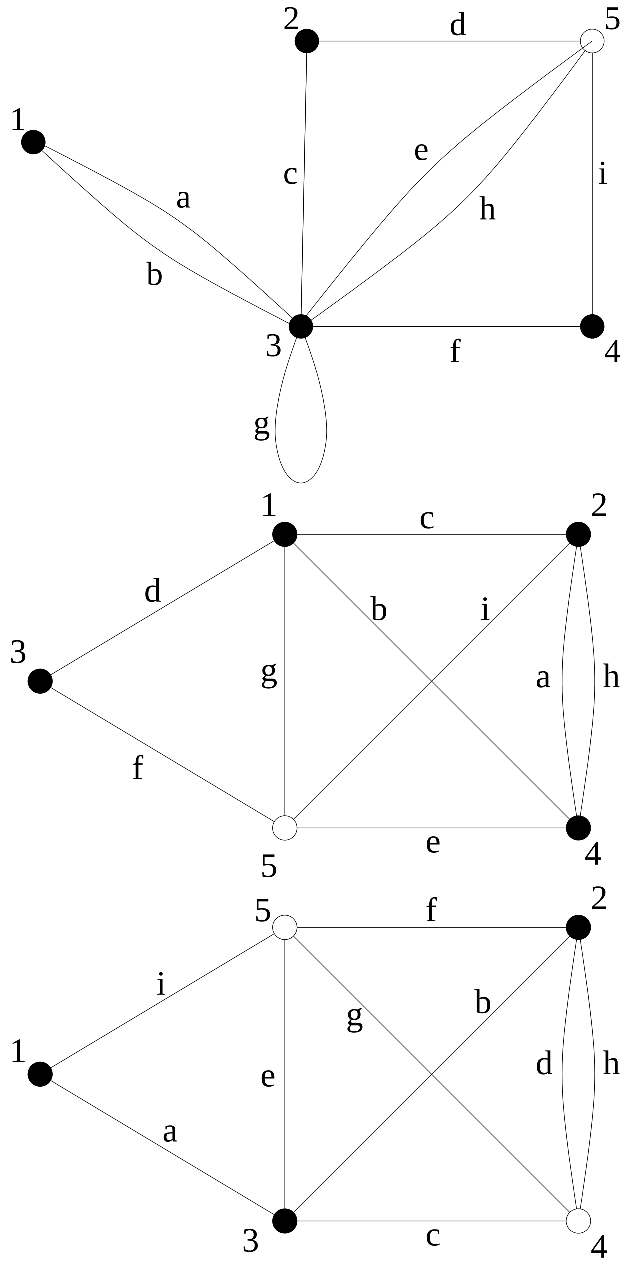


Figure 6: These graphs once again show the respective duals of Figure 4. However, these graphs are shown as non-embedded schematics, rather than embedded structures. Note the differences between the graphs, and again that the third dual possesses only three boundary vertices.

3 The GENUS2$^{\text{TM}}$ Software Package

The GENUS2$^{\text{TM}}$ software package is a set of programs to aid the discovery and construction of graph embeddings of higher genus. Much as Bill Gates purchased DOS from Tim Patterson for mere pennies and made millions off of it, Biesel, Eaton & Associates have inexpensively obtained code from Nick Reickert [3]...and intend to make millions off of it.

The main program is genus2.m. This calls several subprograms, useful in their own right:

- crunch.m
- cyclicperms.m
- read_rot.m
- genus_face.m

The program makefaces.m complements the other programs in the package. Unfortunately, do to current technological limitations in processor speed and virtual memory allocation, the utility of the programs is compromised. We are gratefully seeking any help or advice in how to make said programs run more efficiently.

3.1 The Adjacency Matrix

The main argument of GENUS2() is the 'adjacency matrix'[3]. Each row/column corresponds to a vertex of the graph, with a 1 appearing in the adjacency matrix if there is an edge between two nodes, and a 0 if there is no edge, with 0 entries on the diagonal. Note the zero entries of this will be in the same places as in the Kirchhoff matrix [1], with the added zeros on the diagonal. For example, the adjacency matrix for the 4-3-9 graph in Figure 1 of Section 2 is:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

There is one more entry than there are nodes in Figure 1. This 'eighth' node connects to each boundary node and ensures a cellular embedding as described in Section 2.1.

3.2 genus2.m

GENUS2($adjmatrix$, '$filename'$) is the main program to call to calculate data about graph embeddings. In brief, it checks all possible rotation systems obtained from the adjacency matrix and records data about each. The data is written to a file, and a few basic calculations are printed. The steps followed by GENUS2 are as follows:

- Construct an elementary rotation system matrix L with rotation vertices ordered in ascending order, and zeros filling out the row if the number of columns is greater than the degree of the node.
- Call CRUNCH(L) to construct a matrix *allperms* encoding all possible permutations of the rotation system L.
- Use READ_ROT($rotindex, numvert, allperms$) convert row *rotindex* of *allperms* pack to a rotation system matrix and feed each row to GENUS_FACE($rotsyst$) to find the genus and face degrees of the rotation system *rotsyst*.

- Keep track of faces, row index, and the number of quasi-similar rotation systems to each.
- Print the number of different rotation systems.
- Save row indices, number of quasi-similar embeddings, and faces in array *facetable* and array *allperms* to the file '$filename'$.

3.3 crunch.m

CRUNCH(*rotsyst*) takes any rotation system as the argument and finds all possible permutations of the rotation system. The rotation system should be as described in Section 1, with zero entries filling out any row that corresponds to a node with degree less than the maximum degree in the graph. Accordingly, it is important that no node be labeled as the 0-node, as the program will read this as a non-entry.

$$\begin{bmatrix} 5 & 7 & 8 & 0 \\ 5 & 6 & 8 & 0 \\ 6 & 7 & 8 & 0 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 4 & 0 \\ 2 & 3 & 4 & 0 \\ 1 & 3 & 4 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

CRUNCH encodes these in the array *allperms* with each row being the vector formed by placing the rows of the ith permutation of *rotsyst* one after another. Thus each row represents an entire rotation system, and has length equal to the product of the number of rows and columns of the rotation system matrix, in this case 32. The number of columns will be the number of possible rotation systems, in this case 2304.

Essentially, how CRUNCH works is by calling CYCLICPERMS(row) to compute all of the cyclic permutations of vertices adjacent to a given node, and then finds all possible combinations of these sets of vertices. CYCLICPERMS simply ignores the first entry and any zero entries of a row and uses MATLAB's built in PERMS function to find all of the cyclic permutations of the vertices of a node.

3.4 read_rot.m

READ_ROT(*index*, *vertices*, *permarray*) converts the row *index* of the *permarray* array into a rotation system matrix by dividing the specified index row into a matrix with one row for each vertex.

3.5 genus_face.m

GENUS_FACE(*rotsyst*) accepts a rotation system as its argument and follows the algorithm described in Section 1 of building faces by moving from node to node according to the adjacency specified in the rotation system matrix. It keep track of the degree of each face and the total number of faces to calculate the genus and returns a vector *facelist* with the genus as the first entry, and degrees of each face following in descending order.

3.6 makefaces.m

MAKEFACES(*rotsyst*) reads a rotation system and uses the same method as GENUS_FACE to print the ordering of the vertices in each face in an array. Each row corresponds to one face, with zero entries filling if the degree of the face is less than the maximum degree. The bulk of the code is identical to the GENUS_FACE, but with a tracker for to store the path traversed on each face, rather than a counter for the degree of the faces. Both of these programs differ by less than ten lines from the program GENUSOCE(*rotsyst*) written by Nick Reickert [3].

3.7 An Example of the GENUS2^{TM} Software Package

Here is an example of an application of the GENUS2^{TM} Software Package used to recover the embedding of the 4-3-9 graph with face degrees 8 6 4 4 4 (See the rightmost illustration of Figure 2).

```
>> load variables;     % load the .mat file 4_3_9 adjacency matrix is stored.

>> K        % K is the adjacency matrix for the 4_3_9 graph.

K =

     0     0     0     0     1     0     1     1
     0     0     0     0     1     1     0     1
     0     0     0     0     0     1     1     1
     0     0     0     0     1     1     1     1
     1     1     0     1     0     0     0     0
     0     1     1     1     0     0     0     0
     1     0     1     1     0     0     0     0
     1     1     1     1     0     0     0     0

>> genus2(K, ‘embedK’);     % calculates data about graph embeddings and saves to a file.

Total Distinct Rotation Systems:
        2304

Minimum Genus:
     1

Possible Ways to Embed in That Genus:
   120



>> load embedK      % Load the data file made by GENUS2.

>> facetable    % Display one of arrays from the saved data files of GENUS2.

facetable =

% RowIndex  NumOccur    Faces
     1       132        12     8     6     0     0
     2       720        26     0     0     0     0
     3       324        16     6     4     0     0
     4       108        14     8     4     0     0
    17        90        14     6     6     0     0
    44         6        10     8     8     0     0
    50       348        12    10     4     0     0
    53        78        10    10     6     0     0
    66       378        18     4     4     0     0
    70        30         6     6     6     4     4
   162        54        10     4     4     4     4
   200        36         8     6     4     4     4
```

```
>> rot200 = read_rot(200, 8, allperms)  % converts 200th perm to a rotation system.

rot200 =

     5     8     7     0
     5     8     6     0
     6     8     7     0
     5     6     7     8
     1     2     4     0
     2     3     4     0
     1     4     3     0
     1     2     3     4



 >> makefaces(rot200)   % constructs faces of the embedding from rotation system.

ans =

% each row is traces the vertices of one face.

     1     5     2     8     3     7     0     0
     1     8     2     6     3     8     4     5
     1     7     4     8     0     0     0     0
     2     5     4     6     0     0     0     0
     3     6     4     7     0     0     0     0

% From the rotation system and the faces, it is easy to draw the graph,
% resulting in the third embedding of the 4_3_9 graph in the previous
% section.
```

4 Our First Customer: Adam's Request

We received a request to use our program to find an embedding of the graph:

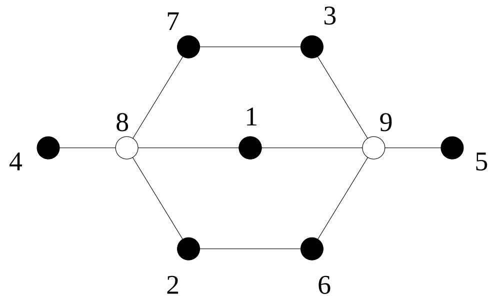


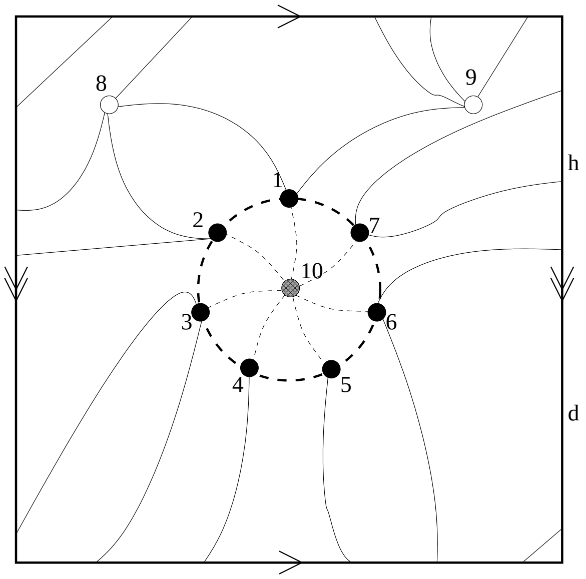
Figure 7: Adam's Requested Graph

Adam wanted to find the minimum circular cellular embedding of this graph, but with the added condition that the boundary nodes be kept in order $1, 2, ...7$ along the boundary circle. After manipulating under 5 lines of code in crunch.m, we were able to produce a modified program which used this restraint to only consider rotation systems that included this ordering (saving time as well as ensuring a correct embedding). Using this we ran the following MATLAB command line with our modified program:

```
>> load adjadam;
>> adjadam

adjadam =

     0     0     0     0     0     0     0     1     1     1
     0     0     0     0     0     1     0     1     0     1
     0     0     0     0     0     0     1     0     1     1
     0     0     0     0     0     0     0     1     0     1
     0     0     0     0     0     0     0     0     1     1
     0     1     0     0     0     0     0     0     1     1
     0     0     1     0     0     0     0     1     0     1
     1     1     0     1     0     0     1     0     0     0
     1     0     1     0     1     1     0     0     0     0
     1     1     1     1     1     1     1     0     0     0

>> genus2adam(adjadam, 'embedadam');
Total Distinct Rotation Systems:
        1152

Minimum Genus:
     1

Possible Ways to Embed in That Genus:
     1

>> load embedadam;
>> facetable

facetable =         % Many lines of facetable are ommitted, but we
  % spot the line indexed by 320 as the one
                    % we desire because it has 7 faces, the proper
  % number for an embedding in genus 1.
      ...
      ...
      ...
      256      4     11      9      5      5      4      0      0
      272      6     11      8      6      5      4      0      0
      290      2     10      8      6      5      5      0      0
      300      2     12      8      6      4      4      0      0
      304      3     15      6      5      4      4      0      0
      320      1      6      5      5      5      5      4      4
      326     12     13     11     10      0      0      0      0
      329     14     26      4      4      0      0      0      0
      335      7     13      8      5      4      4      0      0
      352      2     16      5      5      4      4      0      0
      357     12     16     14      4      0      0      0      0
      ...
      ...
      ...

>> rot320 = read_rot(320, 10, allperms)
```

```
rot320 =

     8    10     9     0     0     0     0
     6    10     8     0     0     0     0
     7    10     9     0     0     0     0
     8    10     0     0     0     0     0
     9    10     0     0     0     0     0
     2    10     9     0     0     0     0
     3    10     8     0     0     0     0
     1     4     7     2     0     0     0
     1     3     6     5     0     0     0
     1     2     3     4     5     6     7

>> makefaces(rot320)

ans =

     1     8     4    10     5     9
     1    10     2     8     0     0
     1     9     3     7    10     0
     2     6    10     7     8     0
     2    10     3     9     6     0
     3    10     4     8     7     0
     5    10     6     9     0     0
```

Using this, we found that there is exactly one embedding of Adam's graph in the torus, and used the rotation system and face system printed to make the following embedding:



A MATLAB Code for GENUS2$^{\text{TM}}$

Here is the MATLAB code for the programs in the GENUS2$^{\text{TM}}$ software package. Very soon these will be obsolete as we intend to replace them with the GENUS3$^{\text{TM}}$ software package coded in java.

A.1 genus2.m

```
function [facetable, allperms]  = 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.
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 CRUNCH to construct an array of all distinct rotation systems of K.
allperms = crunch(L);

% 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(allperms,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, allperms));
    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', 'allperms', 'time');
```

A.2 crunch.m

A.3 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;
```

A.4 genus_face.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];
```

A.5 read_rot.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
```

A.6 makefaces.m

```
function faces = 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'.

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;
            faces(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;
                faces(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
```

References

[1] Curtis, B., and James A. Morrow. “Inverse Problems for Electrical Networks.” Series on applied mathematics – Vol. 13. World Scientific, ©2000.

[2] Esser, Ernie. “On Solving the Inverse Conductivity Problem for Annular Networks.” University of Washington, August 2000.

[3] Reickert, Nick. “Generalized Circular Medial Graphs.” University of Washington, August 2004.