4-19-1994

Average Genus of the Cube

Jody Koenemann
Western Michigan University, jody.koenemann@jackson.com

WESTERN MICHIGAN UNIVERSITY
1903

THE CARL AND WINIFRED LEE HONORS COLLEGE

CERTIFICATE OF ORAL EXAMINATION

Jody Koenemann, having been admitted to the Carl and Winifred Lee Honors College in 1990, successfully presented the Lee Honors College Thesis on April 19, 1994.

The title of the paper is:

"Average Genus of the Cube"

[illegible]

Dr. Arthur T. White
Mathematics & Statistics

Gary Chartrand

Dr. Gary Chartrand
Mathematics & Statistics

[illegible]

Dr. Elise deDonker

The Average Genus of the Cube

Honors Thesis

Jody Koenemann

Winter, 1994

I. Motivations for research

In recent years, there has been interest in the mathematical community in a rapidly developing branch of theoretical mathematics known as random topological graph theory. This new area of mathematics explores the different ways in which certain graphs can be imbedded in given surfaces. The random nature of the new branch results when one also imposes a random distribution on set of all imbeddings of a fixed graph, via the the orientation of the edges at each vertex.

Many interesting results have been discovered in the field of topological graph theory. One that is very interesting is that it was proven mathematically that there exist only five regular polyhedra. For a full explanation and proof, refer to [3]. In addition, graph theory has been utilized to model complex circuits that are being put on computer chips. Thus, the applications that are developing in our technological society are immense.

In 1960, J. Edmonds conducted research on an algorithm to investigate the various ways in which a graph can be imbedded on a surface (see [1]). He documented a very powerful algorithm to explore these imbeddings, commonly named the Edmonds' Permutation Technique.

Using this technique and a random topological graph theory model developed by Lee and White in [2], we shall explore the imbeddings for the graph Q_3 using a particular group and a particular generating set. Hopefully, the results obtained will help later researchers develop a theory by which we can predict the nature of a general cubic graph's imbeddings through knowledge of its rotation scheme.

I have chosen to research random topological graph theory in order to complete my undergraduate honors curriculum by producing an honors thesis. My studies at Western Michigan University have given me the opportunity to study statistics and computer science, but I have been lacking in my development of abstract thought. Before embarking on a career in the very applied field of actuarial science, I believed that it would be helpful to expand my theoretical mathematics knowledge.

Dr. Arthur White has been instrumental in my research. As a highly respected mathematician and researcher of random topological graph theory, Dr. White has done much for the science of mathematics by exploring this new field. I hope that this bit of research that I have conducted will aid him in his research.

II. Definitions

Because of the relative newness of topological graph theory, a few definitions would be in order. The notation discussed below, as found in [2] and [3], will be utilized throughout the paper.

A <u>graph</u> G is a non-empty set of vertices, V(G), together with a set of unordered pairs of distinct veritices, E(G). Each element of E(G) is called an <u>edge</u>. The graph that we shall explore is called Q_3, and it can be pictured as the one-dimensional skeleton of a cube, with eight vertices and twelve edges.

The <u>degree</u> of a vertex v is the number of edges to which v belongs and is denoted by deg(v). The graph Q_3 is <u>cubic</u> in nature, meaning that each vertex v in V(G) has degree three.

A <u>compact</u>, <u>orientable 2-manifold</u> S_k is a topological surface which is topologically equivalent to a sphere with k handles, for $k \geq 0$. For example, S_0 is the sphere and S_1 is the torus (or doughnut).

The graph G with vertex set $V(G)=\{v_1,\ldots,v_m\}$ and edge set $E(G)=\{e_1,\ldots,e_n\}$ is <u>imbedded in a 2-manifold M</u> if there exists a subspace G(M) such that

$$G(M) = \bigcup_{i=1}^{m} v_i(M) \quad \cup \quad \bigcup_{j=1}^{n} e_j(M), \text{ where}$$

(i) $v_1(M),\ldots,v_m(M)$ are m distinct points of M,

(ii) $e_1(M),\ldots,e_n(M)$ are n mutually disjoint open arcs in M,

(iii) $e_j(M) \cap v_i(M)=0$, for $i=1,\ldots,m$ and $j=1,\ldots,n$, and

(iv) if $e_j=(v_{j1},v_{j2})$, then the open arc $e_j(M)$ has $v_{j1}(M)$ and $v_{j2}(M)$ as end points for $j=1,\ldots,n$.

In reference to this definition, an <u>arc</u> is a homeomorphic image of the closed unit interval. An <u>open arc</u> is an arc without its endpoints.

A surface which is topologically equivalent to a sphere with k handles is said to have <u>genus</u> k. The <u>genus of an imbedding</u> of G on S_k is defined to be k. Furthermore, the <u>genus of the graph</u>, denoted $\Upsilon(G)$, is defined as the minimum genus of all such imbeddings of the graph G. If $\Upsilon(G)=k$, an imbedding of G in a surface of genus k is said to be <u>minimal</u>.

The <u>regions</u> or <u>faces</u> of G imbedded on M are the components of the complement of the Image of G on M. With the simple example of the cube which is associated with Q_3, the faces of the graph G would be the six "square" sides of the cube. As in this example, if all the faces are 2-cells (topologically equivalent to open disks), we say the imbedding is a 2-cell imbedding.

A Cayley Graph is a graph G which can be depicted as a set of vertices V(G) and a set of edges E(G) determined as the elements of a finite group Γ. These edges in E(G) depend upon a generating set Δ of the group Γ. In this paper, we describe the set E(G) as:

$$E(G) = \{ \{g, g+d\} \mid g \in \Gamma,\ \delta \in \Delta \}$$

For the Cayley Graph Q_3, the group being used is the group $Z_2 \times Z_2 \times Z_2$, represented by $\{(a,b,c) \mid a,b,c \in Z_2\}$, where $Z_2 = \{0,1\}$, and the generating set $\Delta = \{ (0,0,1), (0,1,0), (1,0,0) \}$. For the sake of clarity, we shall write the elements of this generating set as 001, 010, and 100 respectively.

III. Analytical Techniques.

In order to analyze the nature of the imbedded graph Q_3, it is important to recognize a very useful equation known as Euler's Identity, which applies to every 2-cell imbedding. The equation is

$$F + V = E + 2(1 - k),$$

where F is the number of faces, V is the number of vertices, E is the number of edges, and k is the genus of the imbedding.

Using this simple formula, we can analyze all the 2-cell imbeddings of the graph. Since Q_3 is a graph with eight vertices and twelve edges, we have two of the necessary variables in the equation. In order to obtain the number of faces and hence the value of k, we need to explore a method of examining each imbedding of the graph.

One method of representing 2-cell imbeddings is to utilize an algebraic method known as Edmonds' Permutation Technique. The procedure follows:

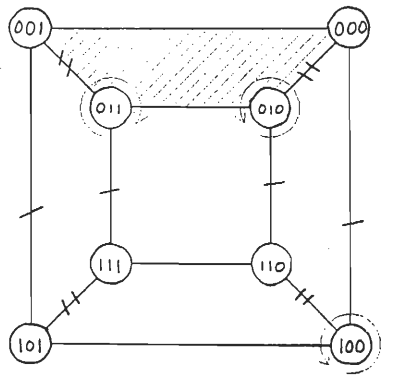
> Suppose a graph G has m vertices; $V(G) = \{1,2,...,m\}$. Let $V(i)=\{k \mid \{i,k\}$ is an element of the set $E(G)\}$. Let p_i be a cyclic permutation of $V(i)$, $i=1,2,...,m$ (of length $m_i = |V(i)|$). The advantages of this representation become apparent when one sees the following theorem.
>
> THEOREM
>
> > Each choice $\varrho=(p_1,p_2,...,p_m)$, called a rotation scheme, determines a 2-cell imbedding G(M) of G in orientable 2-manifold M, such that there is an orientation on M which induces a cyclic ordering of the arcs (i,k) at i in which the immediate successor to (i,k) is $(i,p_i(k))$, $i=1,2,...,m$. In fact, given $(p_1,p_2,...,p_m)$ there is an algorithm which produces the determined imbedding. Conversely, given a 2-cell imbedding G(M) of G in an orientable 2-manifold M with a given orientation, there is a corresponding $(p_1,p_2,...,p_m)$ determining that imbedding. Furthermore, the face boundaries of the imbedding can be computed as follows.

Let $D=\{(a,b)\mid\{a,b\}$ is an element of $E(G)\}$. Define $P:D\rightarrow D$ by : $P((a,b))=(b,p_b(a))$. Then the orbits under the permutation P correspond to the face boundaries.

III. Analysis of the Graph Q_3

Using Edmond's permutation technique, let us work an example dealing with the graph Q_3. Let's take a particular orientation of the edges around each of the eight vertices. For ease of notation, we write each of the vertices as p_i, where i is the value of the binary number corresponding to the group element in the group $\Gamma=Z_2\times Z_2\times Z_2$. According to the notation included above, we have:

$p_0=p_{000}$	:	(001, 100, 010)
$p_1=p_{001}$	:	(000, 011, 101)
$p_2=p_{010}$	:	(000, 110, 011)
$p_3=p_{011}$	:	(001, 010, 111)
$p_4=p_{100}$	:	(000, 101, 110)
$p_5=p_{101}$	:	(001, 111, 100)
$p_6=p_{110}$	:	(010, 100, 111)
$p_7=p_{111}$	:	(011, 110, 101)



Given this set of p_i's, we can determine the face boundaries using the permuation technique. Take the edge going from 000 to 001. The next edge leading away from the vertex 001 can be shown to be (001, p_{001} (000)). Now, p_{001} (000) = 011 since the vertex 011 follows the vertex 000 in the cyclic permutation p_{001}. We can continue this process resulting in the following set of vertices, which are pictured in the drawing above as the shaded region:

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 010 \rightarrow 000 \rightarrow 001$$

Notice that the region ends with a repetition of the first two vertices. Thus, we express the region as the four-sided region:

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 010$$

Similarly, we have the other regions given by:

$$\begin{array}{l} 000 \rightarrow 010 \rightarrow 110 \rightarrow 100 \\ 000 \rightarrow 100 \rightarrow 101 \rightarrow 001 \\ 001 \rightarrow 101 \rightarrow 111 \rightarrow 011 \\ 010 \rightarrow 011 \rightarrow 111 \rightarrow 110 \\ 100 \rightarrow 110 \rightarrow 111 \rightarrow 101 \end{array}$$

From the faces shown, we can solve for the genus random variable using Euler's Identity, with $F = 6$, $V = 8$, and $E = 12$. Thus, we have

$$F + V = E + 2(1 - k) \rightarrow k=0.$$

The complete analysis of Q_3 thus is carried out by changing the set of cyclic permutations p_i, for $i=1,\ldots,m$.

Since it is necessary to change the orientation of the edges at each vertex using this algorithm, another interesting aspect of this problem surfaces. Let us place a probability distribution on the orientation of the edges at each vertex. Since the graph is cubic, there are only two possibilities of an orientation scheme at each vertex. Let's define one possibility as clockwise, and the other as counterclockwise. Furthermore, let the probability of selecting counterclockwise be p, and the probability of selecting clockwise be $1-p = q$. Note that once this is accomplished, one can immediately note that the number of clockwise vertices resembles a binomial distribution, with the number of trials equal to the number of vertices, and the probability of "success" equal to p. Topological Graph Theory analyzes the graph using the uniform case, with $p = 0.5$. Now, we can define a random variable Υ to give the genus of the given imbedding. Mathematically we write,

$$\Upsilon : \Omega \rightarrow \mathbb{N} \cap \{0\} \text{ and}$$

$$\Upsilon(Q_3, \varrho) = k, \text{ if } \varrho \text{ imbeds } Q_3 \text{ on } S_k,$$

where Ω is the samples space containing all possible cyclic permutations for the given graph Q_3. Note that the size of Ω with the graph Q_3 is $2^8 = 256$. This analysis of the random imbeddings of the graph G leads to an instance of a developing branch of mathematics known as Random Topological Graph Theory. In this new area of mathematics, the researcher is interested in the value of the genus random variable, $\Upsilon=k$, rather than in the number of "successes". Currently, a method for determining the relationship between the graph, its number of clockwise vertices, and the genus random variable is being explored by Dr. A. T. White at Western Michigan University.

In the example given above, we can determine the orientation of each of the vertices. Let the term counterclockwise apply to adding the vectors of the generating set in the following order: 001, 010, and then 100. Then the term clockwise refers to the application of the vectors in the generating set in the order: 001, 100, and then 010. Thus, the orientation of each of the vertices in the example above is given as:

000	:	clockwise
001	:	counterclockwise
010	:	counterclockwise
011	:	clockwise
100	:	counterclockwise
101	:	clockwise
110	:	clockwise
111	:	counterclockwise

In analyzing the graph Q_3, it is necessary to look at $|\Omega| = 2^8 = 256$ different imbeddings in order to exhaust all possible orientations. A computer program was developed in January of 1994 to analyze these imbeddings for Q_3. This program utilizes Edmond's Permutation Technique to determine the region boundaries for each imbedding. Then, using Euler's identity, the genus of the imbedding is found for each.

The results obtained by analyzing the data follow, organized in columns by the number of clockwise vertices:

i / k	0	1	2	3	4	5	6	7	8	**Totals**
0	0	0	0	0	2	0	0	0	0	2
1	1	0	0	8	36	8	0	0	1	54
2	0	8	28	48	32	48	28	8	0	200
Totals	1	8	28	56	70	56	28	8	1	256

Notice that the example demonstrated earlier in this paper had four clockwise vertices and a genus of 0. Also notice that the totals along the bottom represent the coefficients in the binomial expansion of the expression below:

$$(1+i)^8 = \Sigma \ (^8{}_i) \ p^i \ (1-p)^{8-i} = (p + (1-p))^8 = 1^8 = 1$$

This is simply a restatement of the binomial theorem for this situation.

Given the uniform case, that is at p=0.5, we can determine the expectation of the genus random variable k, commonly called the average genus of Q_3, by the equation

$$E[k] = ((2)(0) + (54)(1) + (200)(2)) \div 256 \approx 1.7734375$$

In the general case, with a random p, we can also determine E[k] by adding each term of the form:

$$(\text{frequency})(k)(p)^i(1-p)^{8-i}$$

where i is the number of clockwise vertices, k is the genus of the imbedding, and frequency denotes the number of occurrences for the given genus along with the given number of clockwise vertices (i.e., the number from the table above).

After going through the analysis, it was determined that the polynomial to analyze is

$$E[k] = 1 + 8p - 28p^2 + 48p^3 - 70p^4 + 128p^5 - 164p^6 + 104p^7 - 26p^8.$$

Differentiating with respect to p and setting equal to 0, we have

$$0 = 8 - 56p + 144p^2 - 280p^3 + 640p^4 - 984p^5 + 728p^6 - 208p^7.$$

Factoring this equation, we have

$$0 = (2p - 1)(-13p^6 + 39p^5 - 42p^4 + 19p^3 - 8p^2 + 5p - 1).$$

Since the above equation has only one rational root at $p=0.5$, I turned to Maple V to give me the other two symbolic roots:

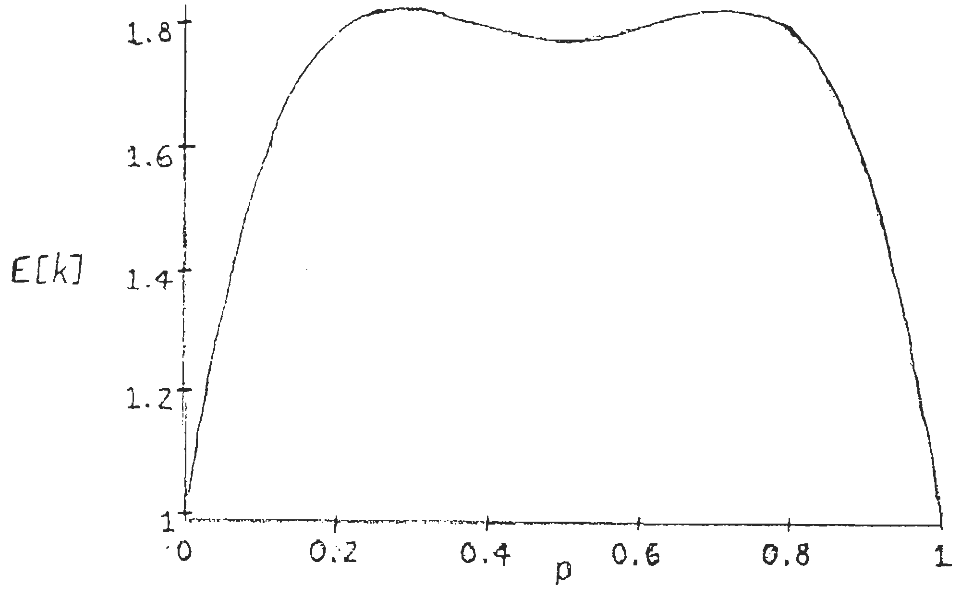
$$0.5 \pm (1/78)(\sqrt{351A^{1/3} - 2028A^{2/3} + 248}\ \sqrt{3}) \div A^{1/6} \approx$$

$$.28315 \quad \text{and} \quad .71685,$$

where $A = (53/2197) + (1/117)\sqrt{11}\sqrt{3}$. The second derivative of the expression was then determined, yielding

$$-56 + 288p - 840p^2 + 2560p^3 - 4920p^4 + 4368p^5 - 1456p^6.$$

Thus, each of the values for p was substituted into the second derivative, indicating whether each was a maximum or minimum value point. The value of E[k] is minimized when $p=0$ and when $p=1$. The two irrational roots mentioned above give the absolute maxima. A relative minimum exists at the point where $p=0.5$ (i.e., in the uniform case). This is an intuitively interesting point since the symmetry about the point $p=0.5$ is a result of the definition used. Namely, if we look to either side of the point where $p=0.5$, it is identical to exchanging each clockwise vertex for counterclockwise, and each counterclockwise vertex for clockwise. These imbedding schema yield reflections of one another, and therefore the same genus value. A graph of the expectation of the genus random variable versus the value of p yields the following:



Also of interest is the value of p that maximizes the expression:

$$Pr[k=0] = 2p^4(1-p)^4 = 2p^4 - 8p^5 + 12p^6 - 8p^7 + 2p^8$$

Setting the first derivative to zero and solving for p, we have:

$$8p^3 - 40p^4 + 72p^5 - 56p^6 + 16p^7 = 8p^3(1-p)^3(1 - 2p)$$

The roots are therefore at p = 0, 1, and 0.5. Checking each of these values using the second derivative, we find that 0 and 1 each yield inflection points which are also absolute minima on the interval [0,1]. The value p=0.5 yields an absolute maximum. Recall that this corresponds to the uniform distribution.

IV. General Summary

We've seen that the rapidly developing field of Random Topological Graph Theory has many interesting questions about one graph Q_3, described by one group $Z_2 \times Z_2 \times Z_2$ and one generating set { 001, 010, 100 }. Based on the data compiled in this study, it seems likely that a relationship can be found between the genus random variable and the particular number of clockwise vertices in the imbedding of the general cubic graph. In addition, it would be interesting to explore whether a relative maximum or a relative minimum of the $E[k]$ can always be found in the uniform case of $p=0.5$. Since we explored the nature of the graph Q_3, it now remains to be seen whether the results for another cubic graph of order eight, say the Cayley graph using the group $\Gamma=Z_8$ and the generators 1 and 4, differ significantly from the results obtained here. Also, what happens when one changes the group for a fixed graph? For example, one could examine the group $\Gamma=Z_2 \times Z_4$ and $\Delta=\{(0,1),(1,0)\}$ or the group $\Gamma=D_4$ (a nonabelian group giving the symmetries of a square) and Δ consisting of a 90° rotation and reflection. Both of these yield the graph Q_3 again. Since a cubic graph necessarily has an even number of vertices, we could then study cubic graph of order 10, 12, 14, and so on. All of these questions are interesting, and they should be examined closely in the future.

BIBLIOGRAPHY

1 J. R. Edmonds, A Combinatorial Representation for Polyhedral Surfaces, Notices Amer. Math. Soc. 7 (1960), 646.

2 Sang H. Lee and Arthur T. White, Random Topological Graph Theory, Proceedings of the Second International Conference in Graph Theory, Combinatorics, Algorithms and Applications (Y. Alavi, F. R. K. Chung, R. L. Graham, and D. F. Hsu, editors), SIAM, 1991, 599-613.

3 Arthur T. White, Graphs, Groups, and Surfaces, Revised Edition, North-Holland, Amsterdam, 1984.

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(*   This program shall analyze the graph Q3, with it's 8 vertices and 12
(*   edges.  The program is modularized extensively, to allow the program to
(*   be read by programmers as well as nonprogrammers.  Each module details
(*   what shall occur within its framework.  The program was written by
(*       Jody Koenemann with assistance from Dr. Arthur White
(*       Western Michigan University
(*       Winter, 1994
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(*  The graph Q3 must be represented in a matrix format as the vertices &
(*  and the vertices that they each connect with by an edge.  A single
(*  embedding is therefore an array of dimensions 8 by 3, since there are
(*  eight vertices and three edges to each vertex.  An example of such an
(*  array follows, followed by whether the vertex is said to be clockwise or
(*  counterclockwise in orientation.
(*
(*  M:        0  1  2                         ORIENTATION
(*          |-          -|                    ===========
(*        0 |  1  4  2   |                    clockwise
(*        1 |  0  3  5   |                    counterclockwise
(*        2 |  3  0  6   |                    counterclockwise
(*        3 |  2  7  1   |                    clockwise
(*        4 |  5  6  0   |                    counterclockwise
(*        5 |  4  1  7   |                    clockwise
(*        6 |  7  2  4   |                    clockwise
(*        7 |  6  5  3   |                    counterclockwise
(*
(*  Analysis of this graph involves finding how many faces there are in the
(*  embedding described by M.  In order to accomplish this, a parallel
(*  array must be kept showing whether an edge has been visited before ( in
(*  the correct order, since this isn't a commutative operation ).  The
(*  parralel array will consist of a matrix of values, which we shall call
(*  VISITED and NOT_VISITED.  This will make the program more readable.
(*  Then, the algorithm for finding the faces shall consist of determining
(*  which edges are still left NOT_VISITED.  Although it would be nice to be
(*  able to examine the nature of the faces, we will not attempt this
(*  immediately.  NOTE:  A possibility for storing the faces would be to
(*  write them to a text file and then read the text file later.   ENDNOTE.
(*  Without further introduction, let us begin the list of algorithms.
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

program ANALYZE_GRAPH (output,outfile);

type
  vertex_orientation_type = (CLOCKWISE,COUNTERCLOCKWISE);
  vtype = array[0..7] of vertex_orientation_type;
  Mrowtype = array [0..2] of integer;
  embedding_representation = array [0..7] of Mrowtype;
  edge_status = (NOT_VISITED,VISITED);
  rowtype = array [0..2] of edge_status;
  parallel_array = array [0..7] of rowtype;
```

```
var
  VertexSet : vtype;
  (* Becuase of difficulty using an array in a for loop, we needed separate
  (* variable names for each vertex.  Once the values were set by the for
  (* for loop, they were then each entered into the array for further
  (* processing. *)
  v0,v1,v2,v3,v4,v5,v6,v7 : vertex_orientation_type;

  M : embedding_representation;
  Number_of_Faces : integer;
  IsVisited : parallel_array;
  counter : integer;  (* counter is used to determine when all of the faces
                      (* are accounted for.  Since the number of edges is 12,
                      (* we know that the product of the number of faces &
                      (* the number of sides per face is 2*12=24. *)
  outfile : text;     (* Used to store the results of the study. *)
  row_index : integer;(* Used to print out the embedding to the file. *)
  genus_array : array [0..2] of integer;
  Number_clockwise, Number_counterclockwise : integer;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(* Update_M_with(VertexSet) is a procedure which uses the VertexSet that
(* describes the orientation of each vertex, and actually updates M with the
(* correct orientation.  The orientation is determined through the use of
(* group theory.  It can be described mathematically as the group Z2*Z2*Z2.
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
procedure Update_M_with (VertexSet:vtype);
  type
    binarytype = 0..1;
  var
    index,indexcopy : integer;
    z1,z2,z3 : binarytype;

  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  (* Orient_cwise takes the binary representation of the vertex and orients
  (* the vertex in a clockwise direction.  Thus, the values 001,100,010 are
  (* added to the vertex in that order.  Index is referred to as a global
  (* variable.
  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
  procedure Orient_cwise(z1,z2,z3: binarytype;
                  var M:embedding_representation);

    begin (* procedure Orient_cwise
      M[index][0]:= z1*4 + z2*2 + (z3+1) mod 2;
      M[index][1]:= ((z1+1) mod 2)*4 + z2*2 + z3;
      M[index][2]:= z1*4 + ((z2+1) mod 2)*2 + z3
    end; (* procedure Orient_cwise

  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  (* Orient_ccwise takes the binary representation of the vertex and orients
  (* the vertex in a counterclockwise direction.  Thus, the values 001,010,
  (* 100 are added to the vertex in that order. Index is referred to as a
  (* global variable. *)
```

```
 (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
 procedure Orient_ccwise(z1,z2,z3: binarytype;
                var M: embedding_representation);

   begin (* procedure Orient_ccwise *)
     M[index][0]:= z1*4 + z2*2 + (z3+1) mod 2;
     M[index][1]:= z1*4 + ((z2+1) mod 2)*2 + z3;
     M[index][2]:= ((z1+1) mod 2)*4 + z2*2 + z3
   end;  (* procedure Orient_ccwise *)

 begin (* procedure Update_M_with(VertexSet) *)
   for index:=0 to 7 do
     begin (* for *)
         indexcopy := index;
         z3 := indexcopy mod 2;
         indexcopy := indexcopy div 2;
         z2 := indexcopy mod 2;
         indexcopy := indexcopy div 2;
         z1 := indexcopy mod 2;
         if VertexSet[index] = CLOCKWISE then
           begin (* if *)
             Orient_cwise(z1,z2,z3,M);
             Number_clockwise := Number_clockwise + 1
           end  (* if *)
         else
           begin (* else *)
             Orient_ccwise(z1,z2,z3,M);
             Number_counterclockwise := Number_counterclockwise + 1
           end;  (* else *)
     end;  (* for *)
 end;  (* procedure Update_M_with(VertexSet) *)


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(* Initialize_IsVisited initializes the array as all NOT_VISITED values.
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
procedure Initialize_IsVisited;
  var
    index1,index2:integer;
  begin (* procedure Initialize_IsVisited *)
    for index1 := 0 to 7 do
      for index2 := 0 to 2 do
          IsVisited[index1][index2] := NOT_VISITED
  end;  (* procedure Initialize IsVisited *)


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(* Go_Around_A_Face is the "meat" of the program.  It uses the matrix
(* representations M and IsVisited to determine what the faces are on the
(* embedding.  First, this procedure determines which edge has not been
(* VISITed.  This will be the starting point.  Next, it goes all the way
(* around the face, counting the number of edges.  As it does this, it marks
(* the IsVisited matrix with the appropriate updates.  In addition, a screen
(* printout is given for each face.  In the future, the outfile data file
(* will be updated using the route around the region.  Finally, when the
(* route around the face has doubled up on itself (it has gone all the way
(* around the region), the number of sides will be displayed and counter
(* will be updated.
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
procedure Go_Around_A_Face(var counter:integer);
 var
   row_index,col_index:integer;
   first_vertex,second_vertex:integer;
   vertex1,vertex2:integer;

 begin (* procedure Go_Around_A_Face *)
   (* First, we must determine what the first edge is going to be. *)
   row_index:=0; col_index:=0;
   while IsVisited[row_index][col_index mod 3]=VISITED do
     begin (* while *)
        col_index:=col_index+1;
        row_index:=col_index div 3
     end;  (* while *)
   col_index := col_index mod 3;

   first_vertex := row_index;
   second_vertex := M[row_index][col_index];

   IsVisited[row_index][col_index] := VISITED;
   vertex1 := first_vertex;
   vertex2 := second_vertex;

   write(outfile, vertex1:2, ' - ', vertex2:2);

   repeat
     (* find the index where the first vertex resides in the second vertex's row *)
     row_index := vertex2;
     col_index := 0;
     while M[row_index][col_index] <> vertex1 do
       col_index := col_index + 1;

     col_index := (col_index + 1) mod 3;
     vertex1 := vertex2;
     vertex2 := M[row_index][col_index];
     IsVisited[row_index][col_index] := VISITED;
     counter:=counter+1;
     write(outfile,' - ',vertex2:2);
   until ( (vertex1=first_vertex) AND (vertex2=second_vertex) );
   writeln(outfile)
 end;  (* procedure Go_Around_A_Face *)


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(* function Some_Edge_Is_NOT_VISITED checks the array IsVisited for some
(* value NOT_VISITED
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
function Some_Edge_Is_NOT_VISITED : boolean;
 var
```

```
   Found_one : boolean;
   row_index,col_index : integer;
 begin (* function Some_Edge_Is_NOT_VISITED *)
   Found_one := FALSE;
   for row_index := 0 to 7 do
     for col_index := 0 to 2 do
        begin (* for *)
          If IsVisited[row_index][col_index] = NOT_VISITED then
            Found_one := TRUE
        end; (* for *)
   Some_Edge_Is_NOT_VISITED := Found_one
 end; (* function Some_Edge_Is_NOT_VISITED *)


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(* The main procedure first initializes the arrays, then produces an
(* orientation, and finally analyzes that orientation. This is repeated for
(* all 256 possible embeddings.
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
begin (* main *)
  Open(outfile, file_name := 'output.file', history := new);
  rewrite(outfile);

  genus_array[0]:=0;
  genus_array[1]:=0;
  genus_array[2]:=0;

  for v0 := CLOCKWISE to COUNTERCLOCKWISE do
    for v1 := CLOCKWISE to COUNTERCLOCKWISE do
      for v2 := CLOCKWISE to COUNTERCLOCKWISE do
        for v3 := CLOCKWISE to COUNTERCLOCKWISE do
          for v4 := CLOCKWISE to COUNTERCLOCKWISE do
            for v5 := CLOCKWISE to COUNTERCLOCKWISE do
              for v6 := CLOCKWISE to COUNTERCLOCKWISE do
                for v7 := CLOCKWISE to COUNTERCLOCKWISE do
                  begin (* for *)
                    VertexSet[0] := v0;
                    VertexSet[1] := v1;
                    VertexSet[2] := v2;
                    VertexSet[3] := v3;
                    VertexSet[4] := v4;
                    VertexSet[5] := v5;
                    VertexSet[6] := v6;
                    VertexSet[7] := v7;

                    Number_clockwise := 0;
                    Number_counterclockwise := 0;

                    writeln(outfile, 'New orientation.');
                    Update_M_with(VertexSet);

                    writeln(outfile,' M: |-       -|');
                    for row_index := 0 to 7 do
                      begin (* for *)
```

```
                 write(outfile, '   ',row_index:1,'| ',
                          M[row_index][0]:1, '  ',
                          M[row_index][1]:1, '  ',
                          M[row_index][2]:1, ' |');
                 if VertexSet[row_index]=CLOCKWISE then
                   writeln(outfile,'         CLOCKWISE')
                 else
                   writeln(outfile,'         COUNTERCLOCKWISE')
                  end;  (* for *)
                writeln(outfile,'    |-        -|');

                Initialize_IsVisited;
                counter := 0;
                Number_of_Faces := 0;
                while counter<24 do
                  begin (* while *)
                 Go_Around_A_Face(counter);
                 Number_of_Faces := Number_of_Faces + 1;
                  end;  (* while *)
                If Some_Edge_Is_NOT_VISITED then
                  writeln(outfile,'Error.  Done with orientation, but not',
                       ' all edges have been visited.');
                writeln(outfile);
                write(outfile,'Then number of CLOCKWISE vertices is ',
                   Number_clockwise:1);
                writeln(outfile,' and the number of COUNTERCLOCKWISE is ',
                   Number_counterclockwise:1);
                writeln(outfile,'The number of Faces is ',
                     Number_of_Faces:2);
                writeln(outfile,'Thus the genus r.v. = ',
                     ((6-Number_of_Faces) div 2):2);
                genus_array[(6-Number_of_Faces) div 2] :=
                  genus_array[(6-Number_of_Faces) div 2] + 1;
                writeln(outfile,'           *****  End of this orientation',
                            ' *****');
                writeln(outfile);  writeln(outfile);  writeln(outfile)
               end;  (* for *)

  writeln(outfile);  writeln(outfile);
  writeln(outfile,'              FINAL REPORT');
  writeln(outfile,'          Genus          Frequency');
  writeln(outfile,'          -----          ---------');
  writeln(outfile,'            0              ',genus_array[0]:3);
  writeln(outfile,'            1              ',genus_array[1]:3);
  writeln(outfile,'            2              ',genus_array[2]:3);

  Close(outfile)
end.  (* main *)
```