HEURISTIC PROGRAMMING AND THE MINIMAL CROSSING PROBLEM

VINCENT J. GROSSO

HEURISTIC PROGRAMMING AND THE MINIMAL CROSSING PROBLEM

by

Vincent J. Grosso

A Thesis submitted to the Faculty of the
College of Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University
Boca Raton, Florida
August, 1970

HEURISTIC PROGRAMMING AND THE MINIMAL CROSSING PROBLEM

by

Vincent J. Grosso

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Frank O. Hadlock, Department of Mathematics and has been approved by the members of his supervisory committee. It was submitted to the faculty of the Collect of Science and was accepted in partial fulfillment of the requirements for the Degree of Master of Science.

SUPERVISORY COMMITTEE:

Frank O. Hadlock
(Thesis Advisor)

John Freeman

[illegible]

M J Deleon

John Freeman
(Chairman, Department of Mathematics)

[illegible]
(Dean, College of Science)

August 26, 1970
(date)

In dedication to my wife, Kathy.

ACKNOWLEDGMENTS

The author wishes to express his appreciation to Dr. Frank O. Hadlock · the introduction to combinatorial mathematics and for his assistance in writ- ȝ this thesis. I would also like to thank Dr. Frederick Hoffman for his guid- ce and advice in connection with this thesis. And a special thanks to Lou ıhnson for her professional concern and consideration in typing this thesis.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

The purpose of this thesis is to investigate the effectiveness of certain heuristic procedures for embedding graphs in the plane so as to minimize the number of crossings. These procedures were developed by Akers and Hadlock (1), at General Electric Electronics Laboratory, while considering possible approaches to the problem of the automated lay-out of integrated electronic circuits. These procedures, described in the thesis, were programmed for the IBM 360/40 and applied to a library of random graphs.

In the interest of being self-contained, we present all necessary definitions and terms as found in (2) and (3).

A graph G is defined abstractly to be an ordered pair (V, E), where V is a set and E is a binary relation on V. The elements in V are called vertices or nodes, and the ordered pairs in E are called the edges of the graph. An edge is said to be incident with the vertices it joins. For example, the edge (a, b) is incident with the vertices a and b.

A finite graph is a graph with a finite number of points. A non-negative integer, called the multiplicity, can be assigned to every ordered pair of vertices.

In general, there may be more than one edge between two vertices. The number of edges between two vertices is the multiplicity of this pair of vertices. Figure A shows a graph where each ordered pair of vertices has a multiplicity greater than one.

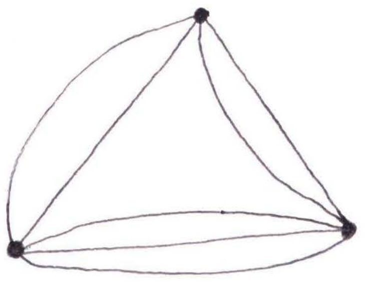

Figure A

A linear graph is one that contains no ordered pair of vertices with multiplicity larger than one.

A directed path of N edges ($E_1, E_2, E_3, \ldots, E_N$) is a sequence of edges such that the terminal vertex of the edge E_i coincides with the initial vertex of the edge E_{i+1} for $1 \leq i \leq N-1$. A directed circuit is a path in which the terminal vertex of E_N coincides with the initial vertex of E_1.

Because our work involves undirected graphs, (graphs where directions are not assigned to the edges), a path (circuit) is defined to be a sequence of edges to which directions can be assigned in such a way that the sequence becomes a directed path (directed circuit). A graph is <u>connected</u> if every pair of vertices are joined by a path.

A <u>tree</u> is a connected graph that contains no circuit. A <u>spanning tree</u> of a graph is a subgraph which is a tree and contains all the vertices in the graph. A graph $G' = (V', E')$ is a subgraph of a graph $G = (V, E)$ if V' is a subset of V and E' is a subset of E. A <u>branch</u> of a tree is an edge that is in the tree. A <u>chord</u> (relative to a tree) is an edge adjacent to the tree but not contained in the tree.

A <u>planar graph</u> is a graph which can be mapped into a plane in such a way that no two edges intersect one another except at a vertex. The intersection of a pair of edges, except at a vertex is called a <u>crossing</u> of the edges. (See example below).

Example I: K_5. In Figure IA we see a mapping of K_5. There are five crossings marked with x's. We can redraw K_5 as in Figure IB. This time there is only one crossing.

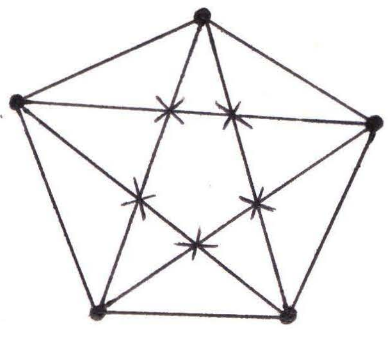
Figure IA

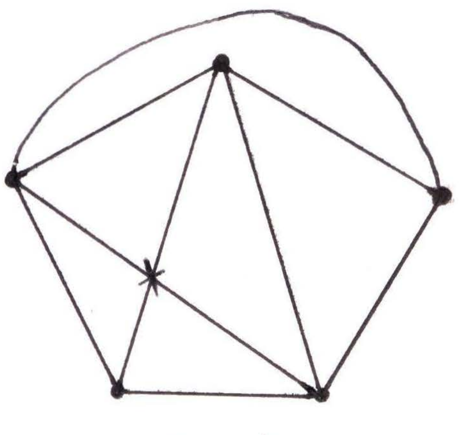
Figure IB

Example 2: $K^{3,3}$. In Figure IC we see a mapping of $K^{3,3}$. There are nine crossings marked with x's. We can redraw $K3,3$ as in Figure ID. This time there is only one crossing.

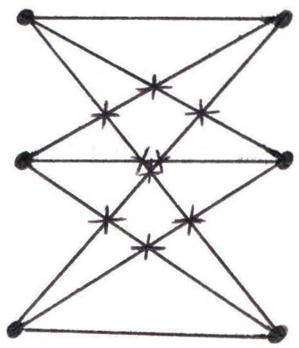
Figure IC

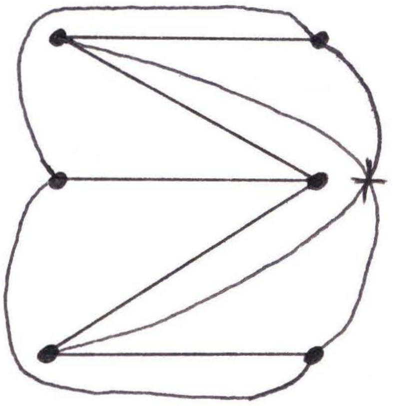
Figure ID

These are examples of the crossing problem. That is, how can we map a given graph in a plane to minimize the number of crossings ? In this thesis we shall create some heuristic procedures to answer this question.

Continuing with our definitions, to lay-out or to map a graph into a plane is simply to draw the graph into the plane subject to these two constraints:

(l) Except at a vertex, a pair of edges may cross each other, at most, one time.

(2) Except at a vertex, no more than two edges may cross at a single point.

A complete graph K_n is defined as a graph in which there is an edge between every pair of the n vertices. A bipartite graph is one whose vertices can be partitioned into two subsets V_1 and V_2; such that every edge in the graph is incident to one of the m vertices in V_1 and one of the n vertices in V_2. A complete bipartite graph $K_{m,n}$ is a bipartite graph such that each vertex in one partition is adjacent to every vertex in the other partition.

In this thesis we shall only concern ourselves with linear, finite, connected graphs.

After programming our heuristic procedures we will examine the output to determine if our procedures have failed. If they fail, we will attempt to correct them. If this is not possible, we hope that our examination has given us an insight for better heuristic procedures.

CHAPTER II

THE RELATED CROSSING PROBLEM AND A COLORATION PROBLEM

We shall discuss a relatively straight-forward procedure for laying out graphs. The layout, will generally involve relatively few, if not the minimum, number of crossings. A graph coloration problem arises in a very natural way from the layout procedure.

Assume for the time being that the graph to be laid out has a Hamiltonian circuit. A Hamiltonian circuit is a circuit that passes through each of the vertices in a graph exactly once. The first step in the procedure is to lay the circuit out with no crossings. The embedded circuit divides the plane into two faces. For any remaining pair of edges (those not in the circuit), it is obvious as to whether they can both be drawn entirely within the same face without intersecting. If not, the two edges will be said to intersect with respect to this particular Hamiltonian circuit. Clearly this is the case if the endpoints of the two edges alternate as the circuit is traversed.

The second step is to partition the remaining edges into two disjoint blocks. The idea is to find a partition resulting in the fewest number of intersecting pairs contained entirely within the same block. The reader will probably have anticipated the final step. All edges within one block are drawn within one face and all edges within the other block are drawn within the other face.

A simple example will serve to illustrate the procedure. Consider $K_{3,3}$ which is displayed in Figure 3A with a Hamiltonian circuit illustrated by the heavy lines. In Figure 3B, the circuit is embedded with no crossings and the remaining three edges are lettered a, b, c and drawn in the same face of the circuit.

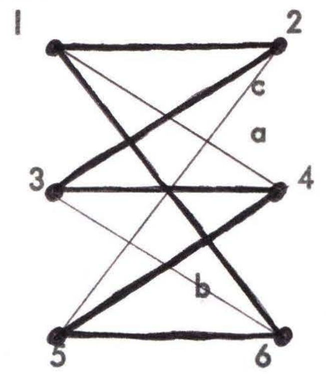

Figure 3A

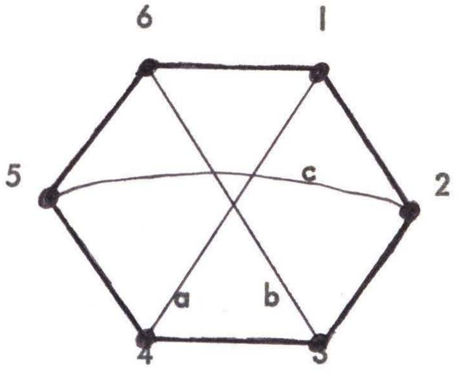

Figure 3B

From this it is seen that every pair of remaining edges intersects with respect to the circuit. There are three two-block partitions of the set $\{a,b,c\}$; each contains one intersecting pair of edges as one block. To each partition corresponds a single crossing layout. To layout $K_{3,3}$ with fewer crossings, a planar embedding would have to be found. But then the Hamiltonian circuit would appear without crossing itself and each remaining branch would be drawn entirely within one face. But these are precisely the conditions we are imposing on the layout. Therefore, for $K_{3,3}$, the procedure yields a minimal crossing layout as displayed in Figure 3C.

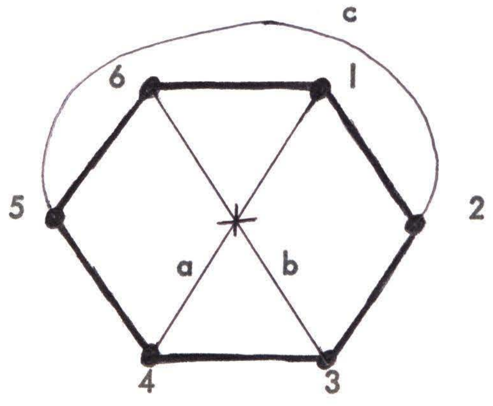

Figure 3C

The procedure will not always yield a minimal crossing layout. The conditions imposed to simplify the problem will sometimes severly constrain the solution. For example, the Mobius Ladder, M_n, (as defined by Guy and Harary) are graphs which can be drawn with far fewer crossing by drawing the Hamiltonian circuit with a crossing. In M_8, Figure 3_D,

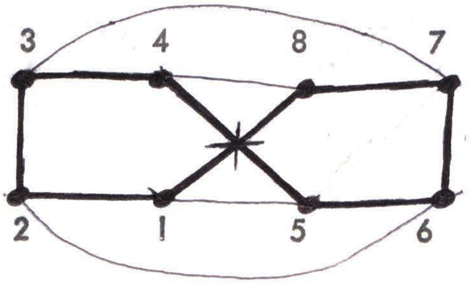

Figure 3 E

there are $n/2 = r$ chords connecting vertices opposite each other (1,r 1), (2, r + 2) ... (r,m). Using our method, the minimal crossing number is far greater than the one crossing we get by having the Hamiltonian circuit crossing itself; see Figure 3E.

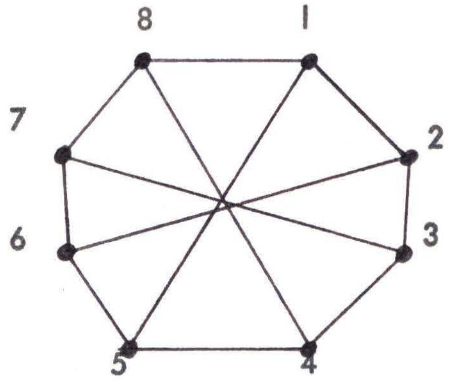

Figure 3D

However, the minimal number of crossings of a large class of graphs can be found by finding the Hamiltonian circuit which does not cross itself and proceding with our heuristic procedures for finding the minimal number of crossings.

In the example used to illustrate the layout procedure, (see Figure 3B), there were only three remaining edges after the Hamiltonian circuit had been embedded in the plane. These were partitioned between the two faces by considering all possible partitions.* As a rule this will be impractical, as

* For additional information the reader is referred to Chapter III, Heuristic Procedure A.

if there are N remaining edges, there will be 2^{N-1} partitions of the edges between the two faces. (No distinction is made between the faces). Therefore, a procedure is needed for selecting a best partition without looking at all partitions. ** It is natural to formulate this as a graph coloration problem.

Given a graph with a Hamiltonian circuit, another graph (hereafter called the "crossing graph" of the remaining edges) is constructed as follows. Represent each remaining edge by a vertex in the crossing graph. Connect two vertices in the crossing graph if the pair of remaining edges they represent happen to intersect with respect to the Hamiltonian circuit*** now the problem of partitioning the remaining edges between the two faces of the Hamiltonian circuit becomes the problem of two-coloring the nodes of the crossing graph so as to minimize the number of edges which have both endpoints colored the same. To see this, let us agree to call an edge of a colored graph either proper or improper, depending on whether the endpoints are colored differently or the same. Then corresponding to an improper edge of the colored crossing graph, we have a pair of the remaining edges of the original graph which intersect with respect to the Hamiltonian circuit and which are to be drawn in the same face of the circuit, thus representing a crossing in the final layout. To minimize the number of crossings, then,

** For additional information the reader is referred to Chapter III, Heuristic Procedure B & C.

*** This construction is similar to that of finding the dual graph of a planar graph. (pg. 113, Harary)(2) we wish to color the crossing graph so as to minimize the number of improper edges.

To illustrate these ideas, consider again the graph $K_{3,3}$. The crossing graph of the three remaining edges is a triangle since each pair intersects with respect to the Hamiltonian circuit. For details examine the sequence of Figures 3A, 3B and 3F.

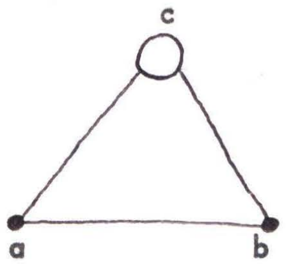

Figure 3F

Since it contains an odd circuit, the crossing graph has no perfect colorations. It can be colored with one improper edge and so, by coloring the crossing graph, we obtain a layout of $K_{3,3}$ with one crossing.

We now see that we can solve the minimal crossing problem by examing the dual problem, that of a best two-coloration. Once achieving this best two-coloration we can transform this result to solve the original minimal crossing problem.

CHAPTER III

MINIMAL VERTEX TWO-COLORATIONS

EXHAUSTIVE GENERATION

Given a crossing graph, we can exhaustively examine every possible vertex two-coloration of the graph. Hence, for a graph of N vertices, there are 2^N possible different colorations. We can then calculate the number of improper edges for each coloration. The program for this heuristic procedure, where x=.5 and the number of vertices is 16, is as follows:

```
      DIMENSION  GRAPH(16,16),JCOLR(16),JGRPH(16,16)
      DIMENSION  NSAMN(16),ITAB(8),KSCLR(16,16)
      REWIND  008
      READ(8) N, ((GRAPH(I,J),J=1,N),I=1,N)
      IF(N,LT,16)  GO TO 1
      X=.5
      NN=N-1
      NBN=0
      NPT=1
      DO 2 I=1,N
      JCOLR(I)=0
      NSAMN(I)=0
      ITAB(I)=400
      DO 2 J=1,N
      KSCLR(I,J)=0
      JGRPH(I,J)=0
      IF(GRAPH(I,J).LE.X)  GO TO 2
      GRAPH(I,J)=3.
      JGRPH(I,J)=1
      CONTINUE
      DO 3 I=1,NN
      II=I+1
      DO 3 J=II,N
      NSAMN(I)=NSAMN(I)+1
      NSAMN(J)=NSAMN(J)+1
      NBN=NBN+1
      NAX=NBN
      ITAB(1)=NBN
```

```
NPT=1
I=1
M=I
DO 5 L=1,N
IF(M-2(M/2).EQ.0)  GO TO 6
M=M/2
K=N-L+L
NACC=O
DO 19 J=1,N
NIND=(1-2*JCOLR(K)-2*JCOLR(J)+4*JCOLR(K)*JCOLR( J))*JGRPH(J,K)
NACC=NACC+NIND
NSAMN(K)=NSAMN(K)-NIND
NSAMN(J)=NSAMN(J)-NIND
CONTINUE
JCOLR(K)=1-JCOLR(K)
NBN=NBN=-NACC
IF(NAX.LE.NBN)  GO TO 25
DO 23 L=1,N
KSCLR(NPT,L)=JCOLR(L)
ITAB(NPT)=NBN
NAX=ITAB(1)
NPT=1
DO 24 L=1,10
IF(ITAB(L).LE.NAX)  GO TO 24
NPT=L
NAX=ITAB(L)
CONTINUE
JJJ=I-2**N+1
IF(0.LE.JJJ)  GO TO 26
I=I+1
GO TO 4
DO 28 L=1,10
WRITE (3,96) ITAB(L),(KSCLR(L,J),J=1,N)
FORMAT(10X,14,10X,20I4)
CALL EXIT
END
```

This program prints out each permutation and the number of improper edges (edges with both endpoints colored the same) for each permutation.

For small N($\leq$ 10) the procedure is very efficient. As N increases, the efficiency of the program is extremely hampered. That is, for a large N, say N=20, we need to examine 2^{20} possible two color permutations of the graph. The computer time and the larger volume of output received make this program unfeasible. Therefore, we must search for a more efficient procedure.

RECOLORING ONE VERTEX AT A TIME

For our second procedure, we initially color each vertex of the crossing graph the same color. We define the excess (possibly negative) of a vertex to be the number of improper edges minus the number of proper edges (edges with different colored endpoints) incident to the vertex. We then change the color of the vertex with the maximum excess. Initially, this vertex is the one with the largest degree. The degree of a vertex is the number of edges incident to the vertex. The result of changing the color of this vertex is that the immediate increase in the amount of proper edges if maximized. That is, we change the largest amount of improper edges to proper edges; while the number of proper edges changed to improper edges is kept at a minimum.

We again calculate the excess of each vertex, changing the color of the vertex with the largest excess. We continue in this manner until the excess of each vertex is a non-positive number. Now, changing the color of any vertex will not improve our two-coloration. For instance, if the maximum excess is less than zero, we have more proper than improper edges incident to each vertex. Changing the color of any vertex will then give more improper edges than the preceding coloration. Hence, we should stop because we cannot improve on our coloration by changing the color of a single vertex.

The program for this heuristic procedure, where X=.5 and the number of vertices is 16, is as follows:

```
      DIMENSION  GRAPH(16,16),ID1R(200,KXSS(200),KEDGE(500)
      DIMENSION  ICOLR(200),NUBCH(200)
      REWIND  008
  1   READ (8) N,((GRAPH(I,J),J=1,N),I=1,N)
      IF(N.LT.16)  GO TO 1
      DO 10 I=1,200
      ICOLR(I)=0
      KXSS(I)=0
      KEDGE(I)=0
      10NUBCH(I)=0
      ID1R(1)=1
      LAST=1
      KPNT=0
      NN=N-1
      X=.5
      DO 2 I=1,NN
      II=I+2
      DO 2 J=II,N
      IF(GRAPH(I,J).LE.X) GO TO 2
      GRAPH(I,J)=3.
      GRAPH(J,I)=3.
      KPNT=KPNT+1
      KXSS(I)=KXSS(I)+1
      KEDGE(KPNT)=J
      IF(I.EQ.LAST) GO TO 2
      IDIR(I)=KPNT
      LAST=I
  2   CONTINUE
      IDIR(N+1)=KEDGE(N)+KXSS(N)
      MAX=KXSS(1)
      MAXPT=1
      DO 4 I=1,N
      IF(KXSS(I).LE.MAX) GO TO 4
      MAX=KXSS(I)
      MAXPT=I
  4   CONTINUE
      IF(MAX.LE.0) GO TO 7
      ICOLR(MAXPT)= 1-ICOLR(MAXPT)
      NUBSCH(MAXPT)=NUBCH(MAXPT)+1
      L=ID1R(MAXPT)
      M=ID1R(MAXPT+1)-1
      DO 6 K=L,M
      MM=KEDGE(K)
      IF(ICOLR(MAXPT).EQ.ICOLR(MM))   GO TO 5
      KXSS(MAXPT)=KXSS(MAXPT)-2
      KXSS(MM)=KXSS(MM)-2
      GO TO 6
 5    KXSS(MAXPT)=KXSS(MAXPT)+2
      KXSS(MM)=KXSS(MM)+-
 6    CONTINUE
      GO TO 3
 7    CONTINUE
      WRITE(3,8) (ICOLR(I),I=1,N)
 8    FORMAT(1H _,20I4)
      WRITE(3,9) (NUBCH(I),I=1,N)
 9    FORMAT(1H__,20I4)
      CALL EXIT
      END
```

This program should print out a minimal coloration and the number of times each vertex changed colors. However, after examining a large quantity of output we see that in certain specific cases our procedure fails. The details of these failures shall be discussed in Chapter Five. Therefore, we must again create another procedure for finding the best two-coloration of a graph.

CUT-SET METHOD

In the previous sections, we posed the problem of two-coloring the vertices of a graph so as to minimize the number if improper edges. It is known (Konig$^{(2)}$) that a proper coloration exists if and only if the graph contains no odd circuits.

For graphs with odd circuits, a minimal coloration will be one for which the number of improper edges is a minimum. The following theorem charaterizes minimal colorations:

Theorem: Let G be a graph. If C is a two-coloration of the vertices of G, C is minimal if and only if every cut-set contains as many proper as improper edges.

A maximal connected subgraph of a graph is called a connected component or simply component. A cut-set is a (minimal) set of edges in a graph, the removal of which will increase the number of connected components in the remaining subgraph, whereas the removal of any proper subset of which will not. It follows that in a connected graph, the removal of a cut-set will separate the graph into two disjoint, connected subgraphs. A second definition of a cut-set is the set of edges with the endpoints of each edge in opposite components of a two-component partition of the set of vertices.

PROOF: The necessity is immediate. If there exists a cut-set wi th more improper than proper edges,complement the color of every vertex in one of the cut-set's two components. Only edges in the cut-set change status. In the cut-set, improper edges become proper and vice-versa. Hence, the resulting coloration has fewer improper edges. To show the sufficiency, consider a coloration for which no cut-set contains more improper than proper edges. To prove it is a minimal coloration, consider a second coloration. The two colorations induce a cut-set as follows. Partition the vertices into two blocks: Those vertices colored the same under both colorations and those colored differently. We must show that if the second coloration had fewer improper edges than the first, then the cut-set would contain more improper than proper edges, a contradiction. Edges for which the two colorations agree on both endpoints have the same status under either coloration. Edges for which the two colorations disagree on both endpoints have the same status under either coloration. Edges in the cut-set have opposite status. Thus edges in the cut-set which are proper under the first coloration are improper under the second coloration. Hence they are fewer in number than the remaining edges which are improper under the original coloration. But this is a contradiction and so the original coloration is minimal. (I)

This theorem not only characterizes the minimal colorations, it provides an approach for a heuristic scheme for finding them. Obviously we would like to find a cut-set with more improper than proper edges. By complementing all vertices in one component, we reduce the number of improper edges by their excess over the proper edges in the cut-set.

What we propose is a rule for improving a cut-set by moving a single vertex from one side to the other. First define the _excess of a cut-set_ as being the number of improper minus the number of proper edges contained in the cut-set. A cut-set is better than another if its excess is greater.

Now consider the effect on the excess of moving one vertex from one side of the cut-set to the other. The net effect on the excess of moving a vertex will be called the _increment_ of that vertex. The increment, will be the increase in improper edges in the cut-set, plus the decrease in proper edges in the cut-set. Considering only edges incident with the particular vertex, the increase in improper edges will be the number not in the cut-set minus the number in the cut-set. The decrease in proper edges will be the number in the cut-set minus the number not in the cut-set.

The rule is to increase the excess of cut-set by changing the vertex with maximum positive increment until no vertex has positive increment. The final cut-set, if it has positive excess, is used to obtain a better coloration.

The following figures illustrate the procedure:

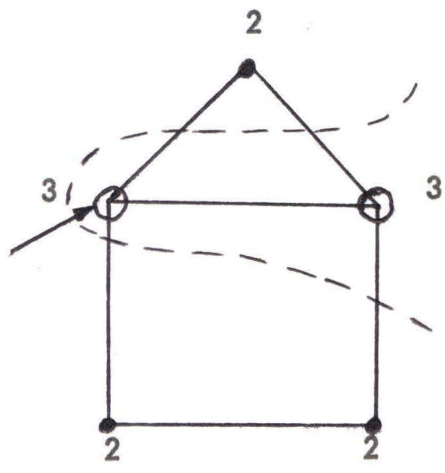

Figure 3Cl

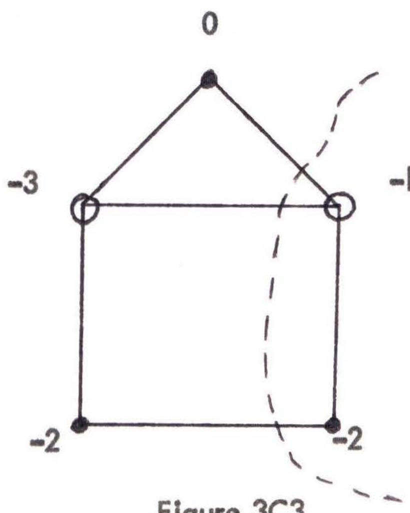

Figure 3C3

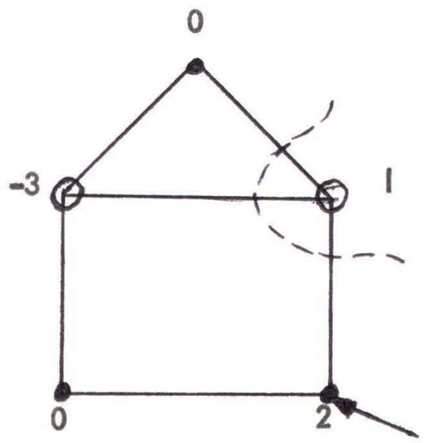

Figure 3C2

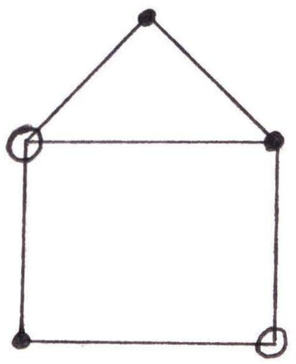

Figure 3C4

In each of the Figures 3Cl, and 3C2 and 3C3 the coloration remains the same. In Figure 3Cl, corresponding to the coloration and indicated cut-set, the increment for each vertex and the excess of the cut-set have been computed. The vertex in Figure 3Cl with the maximum positive increment is indicated by an arrow. This vertex will be moved across the cut-set resulting in a new set of increments for each vertex and a change in the excess of the cut-set. Our graph now has the form of Figure 3C2. The cut-set in Figure 3C2 is obtained from the cut-set in Figure 3Cl by moving the indicated vertex.

Since there still exists a vertex with positive increment, we continue the process to get the graph in Figure 3C3. Because no vertex has positive increment, no further improvement in the coloration can be made. The colors of the vertices in either component are then complimented. This results in a minimal coloration as shown in Figure 3C4.

The program for this heuristic procedure, where X=.5 and the number of vertices of the graph is 16, is as follows:

```
      DIMENSION GRAPH(16,16),KGRPH(16,16),ICOLR(100),ICP(100)
      DIMENSION KTGPH(16,16),JNC(16,16),INC(16),NUMB(100)
      REWIND  008
    1 READ (8) N,((GRAPH(I,J),J=1,N),I=1,N)
      IF(N,LT,16) GO TO 1
      X=.5
      NN=N-1
      DO 2 I=1,N
      NUMB(I)=0
      DO 2 J=1,N
      KGRPH(I,J)=0
      KTGPH(I,J)=0
    2 JNC(I,J)=0
      DO c I=1,NN
      II=I+1
      DO 3 J=II,N
      IF (GRAPH (I,J).NE.3.) GO TO 3
      KGRPH(I,J)=2
      KGRPH(J,I)=2
    3 CONTINUE
      NNN=N-1
      DO 4 I=I,NNN
      III=I+1
      DO 4 J=III,N
      IF(KGRPH(I,J).NE.2) GO TO 6
      IF(ICOLR(I).EQ.ICOLR(J)) GO TO 5
      KTGPH(I,J)=1
      KTGPH(J,I)=1
      KTDM=KTDM-1
      GO TO 4
    5 KTGPH(I,J)=-1
      KTGPH(J,I)=-1
      KTDM=KTDM=1
    6 CONTINUE
      IF(KGRPH(I,J).NE.1) GO TO 4
      IF(ICOLR(I).EQ.ICOLR(J)) GO TO 7
      KTGPH(I,J)=-1
      KTGPH(J,I)=-1
      GO TO 4
    7 KTGPH(I,J)=1
      KTGPH(J,I)=1
      CONTINUE
      DO 8 I=1,N
      DO 8 J=1,N
      JNC(I,J)=KTGPH(I,J)
    8 CONTINUE
      DO 9 I=1,N
      DO 9 J=2,N
      JJJ=J-1
      JNC(I,J)=JNC(I,J)+JNC(I,JJJ)
      INC(I)=JNC(I,J)
    9 CONTINUE
      LMAX=INC(1)
      LPT=1
      DO 10 I=1,N
      IF(INC(I).LE.LMAX) GO TO 10
      LMAX=INC(I)
      LPT=I
   10 CONTINUE
      IF(LMAX.LE.)) GO TO 13
      ICP(LPT)=1=ICP(LPT)
      NUMB(LPT)=NUMB(LPT)+1
      DO 12 J=1,N
      IF(KGRPH(LPT,J).NE.2) GO TO 11
      KGRPH(LPT,J)=KGRPH(LPT,J)-1
      KGRPH(J,LPT)=KGRPH(J,LPT.)=1
      GO TO 12
   11 IF(KGRPH(LPT,J).NE.1) GO TO 12
      KGRPH(LPT,J)=KGRPH(LPT,J)+1
      KGRPH(J,LPT)=KGRPH(J,LPT)+1
   12 CONTINUE
      GO TO 3
   13 CONTINUE
      DO 14 I=1,N
      IF(ICP(I).EQ.)) GO TO 14
      ICOLR(I)=1-ICOLR(I)
   14 CONTINUE
      WRITE(3,97) (ICOLR(I), I=1,N)
   97 FORMAT(1H,20I4)
      WRITE(3,98) (NUMB(I),I=1,N)
   98 FORMAT(1H,'NO. TIMES ACROSS CUT',4X,20I4
       CALL EXIT
       END
```

CHAPTER IV

CREATING A LIBRARY OF RANDOM GRAPHS

Realizing the fact that much of our work is to be done on a computer, we construct a library of random graphs and store them on magnetic tape. This tape is to be a reference library of graphs which we can work from.

To do this, we start with a complete graph of 10 vertices. We then arbitrarily assign random lengths, of up to four decimal places, between 0 and 1 to each edge of the graph. We find the minimal spanning tree and then store the tree and the random lengths of the remaining edges of the complete graph on magnetic tape.

The following is a listing of the program for making a library of random graphs:

```
      DIMENSION IPT(100),JPT(100),GRAPH(100,100)
      IX=13107
      DO 1 N=10,100,1
      AMIN=2.
      DO 2 I=1,N
      IPT(I)=0
      JPT(I)=0
      GRAPH(I,I)=3.
    2 CONTINUE
      NNN=N-1
      DO 3 I=1,NNN
      III=I+1
      DO 3 J=III,N
      IY=IX*65539
      IF(O.LE.IY) GO TO 5
      IY=IY+2**31
    5 YFL=IY
```

```
      YFL=YFL*2.**(-31)
      IX=IY
      KY=IY
      GRAPH(I,J)=YFL
      GRAPH(J,I)=GRAPH(I,J)
      IF(AMIN.LE.GRAPH(I,J)) GO TO 3
      AMIN=GRAPH(I,J)
      IPT(1)=I
      IPT(2)=J
    3 CONTINUE
      GRAPH(IPT(1),IPT(2))=3.
      GRAPH(IPT(2),IPT(1))=3.
      JPT(2)=IPT(1)
      DO 8 KK=3,N
      BMIN=GRAPH(1,1)
      DO 7 I=1,N
      DO 7 L=1,N
      IF(I.NE.IPT(L)) GO TO 7
      DO 7 J=1,N
      IF(BMIN.LE.GRAPH(I,J)) GO TO 7
      DO 6 K=1,N
      IF(IPT(K).NE.J) GO TO 6
      GO TO 7
    6 CONTINUE
      BMIN=GRAPH(I,J)
      KPT=I
      MPT=J
    7 CONTINUE
      JPT(KK)=KPT
      IPT(KK)=MPT
      GRAPH(JPT(KK),IPT(KK))=3.
      GRAPH(IPT(KK),JPT(KK))=3.
    8 CONTINUE
      WRITE (8) N,((GRAPH(I,J),J=1,N),I=1,N)
    1 CONTINUE
      CALL EXIT
      END
```

We store on tape the matrix representation of the complete graph including the minimal spanning tree. Repeating the process, we create random graphs of 11, 12, 13... until we finally stop at the complete graph of 90 vertices.

Now, for example, if we want to work with the family of graphs having 16 vertices we first read the stored graph of 16 vertices.

We will be given a complete graph of 16 vertices with a minimal spanning tree embedded in the graph. Since each chord of the graph has a random length assigned to it, we can arbitrarily add branches to the spanning tree. We pick some number X, where $0<X<1$, so that all the chords with length $\geq X$ will be added to the minimal spanning tree.

We now have an easily accessible library of random graphs from which we can work.

CHAPTER V

RESULTS AND CONCLUSIONS

It is intended to program a technique for finding Hamiltonian circuits in order to apply the lay-out procedure to the random connected graphs in the library. Also, the lay-out procedure will be applied to complete and bipartite graphs (where a Hamiltonian circuit is immediate) and the results compared with available upper bounds. For the present, the results for the coloration procedure as applied to the random graphs from N= 10 to 18 using a threshold of .25 is as follows:

TABLE 5A

	Fewest # of improper edges			Absolute Efficiency		#of Edges
N	A	B	C	A/B	A/C	
10	16	21	17	.875	.975	40
11	20	25	23	.9	.94	51
12	22	24	24	.965	.965	58
13	28	36	33	.884	.928	69
14	31	39	36	.9	.937	79
.						
.						
.						
18	55	67	66	.909	.917	132

More graphs would have been examined if more computer time were available. The time element involved for the graph of 18 vertices was sixty-five minutes and for a graph of nineteen vertices the time would be approximately doubled. A measure of the efficiency of one procedure over another would be one minus the difference between the fewest number of improper edges from one procedure and the fewest number of improper edges from the other procedure, the difference being divided by the number of edges in the graph. The efficiency of procedure A over procedure B is in the fifth column of Table 5A. The efficiency of procedure A over procedure C is in the sixth column of Table 5A.

To determine Table 5A the three heuristic procedures were combined into one program. The program was run from the tape which contained the library of random graphs. The program was started with a threshold of .25 and N (the number of vertices equal to 10). All the branches of lengths greater than .25 were added to the spanning tree. The number of improper edges from procedure A was calculated and compared with the number of improper edges from procedures B and C. The reason for doing this is to find the improvement of one procedure over another.

This procedure was repeated for graphs having 11 vertices through graphs having 18 vertices. Examining the results of these programs it is found that in certain cases procedure B fails. An example of where it fails is as follows:

Starting with a coloration as in Figure 5B the excess of each vertex is a non-positive number this implies that this coloration has the fewest number of improper edges. But clearly, the coloration of the graph in Figure 5C has fewer improper edges.

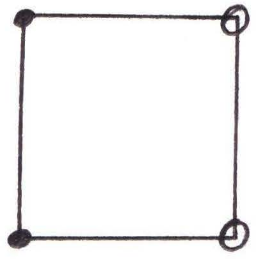
Figure 5B

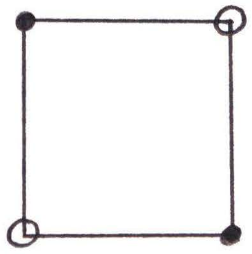
Figure 5C

Hence, changing the color of one vertex does not always get the best two-coloration of a graph. Whereas simultaneously changing the color of two or more vertices results in a better two-coloration of a graph.

Upon further examination of the output it is found that procedure C fails as in the following case:

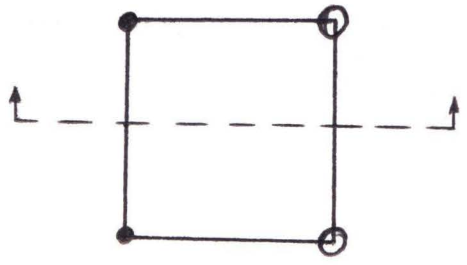

Figure 5D

Starting with the cut-set as in Figure 5D it is seen that every vertex has a non-positive increment. This implies that this coloration has the fewest number of improper edges. But clearly, the coloration of the graph in Figure 5C has fewer improper edges.

Although these procedures fail in certain areas they still give us a fairly good approximation of the fewest number of improper edges of any given graph.

HEURISTIC PROGRAMMING AND THE MINIMAL CROSSING PROBLEM

by

Vincent J. Grosso

An Abstract submitted to the Faculty of the
College of Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University
Boca Raton, Florida
August, 1970

ABSTRACT

There are several theorems which give the upper bounds on the number of crossings of a graph in a plane. In this thesis we shall program certain heuristic procedures for finding the layout of the graph with the fewest number of crossings. We will then examine the output of these procedures to see if they always give us a graph with the fewest number of crossings.

BIBLIOGRAPHY

(1) Akers, Sheldon B.; Hadlock, Frank O. Electrical Networks and the Minimal Crossing Problem. Presented at the Graph Theory and Computing Conference. University of West Indies: Kingston, Jamaica, January, 1969.

(2) Liu, C. L. Introduction to Combinatorial Mathematics. New York, New York: McGraw - Hill Book Company, 1968.

(3) Harary, Frank. Graph Theory. Reading, Massachusetts: Addison-Wesley Publishing Company, 1969

(4) Berge, Claude. The Theory of Graphs. New York, New York: John Wiley & Sons, Inc., 1962.