A Graphical User Interface for the Creation, Manipulation, and Calculation of Electrical Networks

Mark Hoefer

August 9, 1996

1 Introduction

This paper will discuss the creation of a graphical user interface (GUI) for dealing with electrical networks implemented in the Java programming language. This language facilitates applets which are programs that can be run using a World Wide Web browser. Thus, my hope is to provide an easy to use program which allows anyone with web access the opportunity to experience a sampling of the problems that are being discussed in the REU summer program here at the University of Washington.

Topics covered in this paper include: the computer science used to implement the problems, the problems themselves, the GUI created in Java and how to use it, and finally the Java code.

In particular, the applet implemented in Java will be an interface to any user on the Web. One can enter a network (see definition below) by clicking with the mouse on the screen, generating a rectangular lattice network of a specified size or generating a circular network. Then various operations can be performed such as: displaying a $\gamma-$*harmonic* potential function v given boundary voltages and altering the network as desired. These operations will allow the user to get a hands-on graphical demonstration of electrical networks.

2 Electrical Networks

An **electrical network** Ω consists of a graph Γ and a function γ. The graph consists of a set of **nodes** (denoted V) and a set of **edges** (denoted E) which connect nodes. The set of nodes is further divided up into a set of interior nodes (denoted int-Γ) and a set of boundary nodes (denoted $\partial\Gamma$). The function γ associates a number to each edge pq $\in E$, the **conductance** between two nodes. Define a function $v(p)$ to be the electric potential at node p and another function $I(pq)$ as the current flowing across the edge pq. Allow p$\sim$q to mean q is a *neighbor* of p. By this I mean that p and q are connected by only one edge.

By **Ohm's Law** the relation $I(pq) = \gamma(pq)(v(p) - v(q))$ holds. If p $\in$ int-Γ, by **Kirchhoff's Law** notice the following summation:

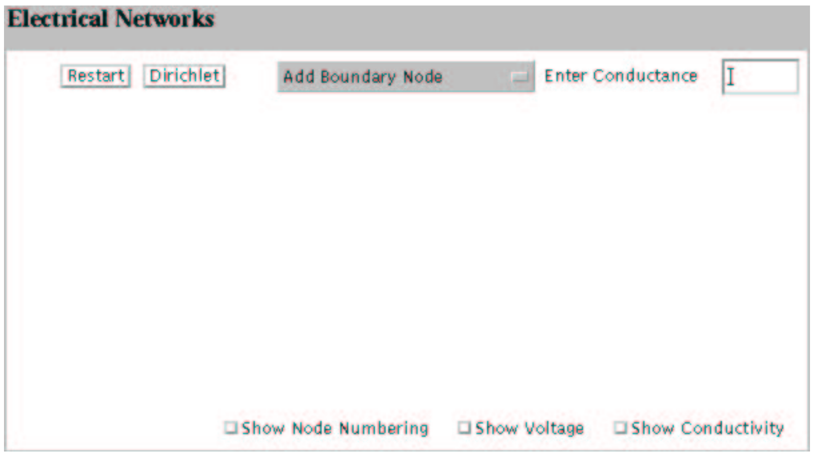
$$\sum_{q,p\sim q} I(pq) = \sum_{q,p\sim q} \gamma(pq)(v(p)-v(q)) = 0$$

A function $v(p)$ which satisfies the above equation for all p $\in$ int-Γ is said to be $\gamma - harmonic$ or to have the *averaging* property.

3 Using the Java Applet

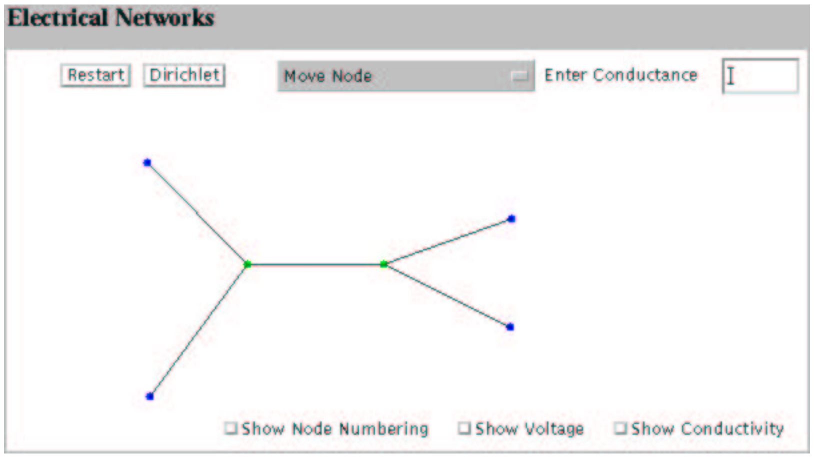
In order to invoke the program, the user must have access to the World Wide Web and a Java enabled browser such as Netscape 2.0 or greater. Now access the URL http://www.math.washington.edu/~morrow/reu96/reu.html and choose the numbered item associated with the DrawNetwork web page.

3.1 Entering a Network



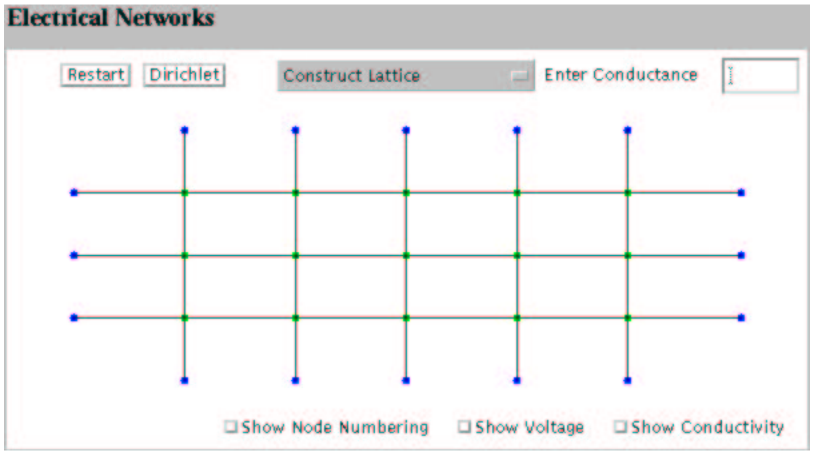
The DrawNetwork Applet

There are several ways in which the user can enter an electrical network. In general, the first step is to choose "Enter Boundary Node" from the drop down menu. Now click on the screen, creating blue boundary nodes. These nodes will be stored in the order in which they are entered. After all boundary nodes have been entered, choose "Enter Interior Node" from the drop down menu. Again, click on the screen, creating green interior nodes. After one interior node has been entered, boundary nodes can no longer be entered. It has to do with the way that the nodes are stored in memory. Once satisfied, connections can now be made. Choose "Set Conductance" from the drop down menu. Notice in the upper right hand corner there is a box with a label "Enter Conductance" to the left of it. Before making a connection between two nodes, highlight the box and enter the desired conductance, notice it defaults to one. Now, click on any two nodes where a connection is desired. A line will be drawn between those two nodes. Now continue making connections and changing the conductivity if desired until the network is constructed. Voila, an electrical network.



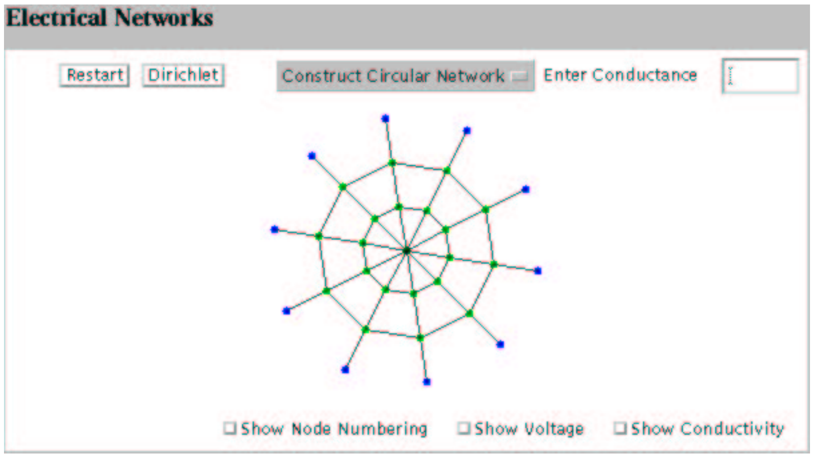
Example Electrical Network

An alternative method of creating an electrical network is by choosing "Construct Lattice" from the drop down menu. A pop up box will appear, prompting the user to enter the number of rows and columns desired. Enter both numbers in their respective boxes and click on "OK". A rectangular lattice will be drawn filling up most of the applet's drawing area. More interior nodes can be added if desired, but not boundary nodes. More connections can also be made. The default conductivity for each connection is one. An individual connection's conductivity can be changed by simply choosing "Set Conductivity" from the menu, entering the conductance as before, and then clicking on the desired two nodes. The largest lattice that is allowable is 15 rows by 15 columns. This is due to the limit on the number of nodes, which is 299.



Example Rectangular Lattice Network (3 x 5)

Another type of electrical network that is easy to construct is the circular network. Choose "Construct Circular Network" from the menu and a pop up box will appear, prompting the user to enter the desired number of boundary nodes and number of radial levels. Remember that the largest number of allowable nodes is 299. Again, the default conductivity is one, yet it can be altered by the same method used in the previous two examples.



Example Circular Network

Once the network has been entered, there are some adjustments that can be made. Of course if all is hopeless, click on "Restart" and the screen will be cleared, allowing the user to start over again. If a node is not in the desired position, then choose "Move Node" from the menu and click on the desired node. Unfortunately, if there are a large number of nodes on the screen, this operation may take a little while due to the algorithm which must check all nodes until it finds the one which was clicked. An edge can be removed by choosing "Remove Connection" from the menu and clicking on the two nodes whose connection must be removed. A node can be removed similarly by choosing "Remove a Node" from the menu and then clicking on the desired node. The nodes will be re-numbered appropriately. There are three pieces of information about the network which can be displayed at any time. Click the "Show Node Numbering" box at the bottom of the screen in order to display the ordering with which the nodes were entered. This will be useful later when the boundary voltages are entered. Choose the "Show Voltage" box and the voltages will be displayed. And finally the "Show Conductivity" box will display the conductivity for each edge. I tend to leave the boxes unchecked because the network becomes very cluttered with information otherwise.

3.2 Dirichlet Problem

I will be concerned with the forward or **Dirichlet Problem**. The problem is stated as follows: given a potential function ϕ defined everywhere on $\partial\Gamma$, what is the $\gamma - harmonic$ function v which satisfies $v(p) = \phi(p)$ for all $p \in \partial\Gamma$? It is shown by [3] that there indeed is a solution and it is unique. This problem can be solved directly by a system of equations.

3.2.1 Obtaining the Solution

In order to solve the Dirichlet problem for the electrical network on screen, the boundary voltages need to be entered. First choose "Show Node Numbering". Now choose "Enter Boundary Voltages" from the drop down menu. A pop up box will appear asking the user to enter the boundary voltage for Node 0. Highlight the text in the box and enter the desired number, now press return. The box now asks the user for the voltage for Node 1. Type in the desired amount and press return. Continue this process until all boundary voltages have been entered. Notice that the node number displayed can be found on the network since that information is currently being displayed. Unclick the "Show Node Numbering" choice and click on "Show Voltage". The boundary voltages will be displayed. Now the solution to the Dirichlet problem can be calculated. Simply press the "Dirichlet" button at the top of the applet and the interior voltages will appear. Changes in the network can be made such as adding more interior nodes, changing conductivities, or changing the boundary voltages. Once these changes have been made, the Dirichlet solution for the new network can be found by simply clicking the "Dirichlet" button again.

3.2.2 The Program's Algorithm

Suppose Ω is a connected electrical network with n nodes numbered $x_1, ..., x_n$. The Kirchhoff matrix K associated with Ω is constructed as follows.

(i) If $i \neq j$ then $K_{i,j} = -\sum \gamma_{ij}$ where we sum the conductance of all edges directly connecting x_i and x_j. Thus if there is only one edge connecting x_i and x_j then $K_{i,j} = -\gamma_{ij}$. Or if x_i is not a neighbor of x_j then $K_{i,j} = 0$.

(ii) $K_{i,i} = \sum_{x_i \sim x_j} \gamma_{ij}$, where we sum the conductance of all edges emanating from x_i.

The Kirchhoff matrix is a mapping from voltages to currents. If we give a special ordering to the nodes $x_1, ..., x_n$ where the first sequence of nodes $x_1, ..., x_d$ are all of the boundary nodes and $x_{d+1}, ..., x_n$ are all of the interior nodes, then the Kirchhoff matrix associated with these nodes can be written as follows.

$$K = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$$

Given a potential function ϕ defined everywhere on the boundary given by

$$\phi = \begin{bmatrix} \phi_1 \\ . \\ . \\ . \\ \phi_d \end{bmatrix}$$

we wish to find the function u defined everywhere on the interior given by

$$u = \begin{bmatrix} u_{d+1} \\ . \\ . \\ . \\ u_n \end{bmatrix}$$

If we append u to ϕ, then we have a $\gamma - harmonic$ function

$$v = \begin{bmatrix} \phi \\ u \end{bmatrix}$$

which is the unique solution to the Dirichlet problem.

Remembering that K takes voltages to currents, we can multiply K by the $\gamma - harmonic$ function v and get the following current.

$$Kv = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} \phi \\ u \end{bmatrix} = \begin{bmatrix} ? \\ 0 \end{bmatrix}$$

The lower portion of the current vector is zero because there are no sources of current in the interior of the network. Doing matrix block multiplication we have

$$B^T\phi + Cu = 0 \;\; or \;\; Cu = -B^T\phi$$

Let $y = -B^T\phi$ then we wish to solve the linear system $Cu = y$. In [3], they showed that C is a symmetric, positive definite matrix. Choleski factorization can now be used on C thus giving $C = LL^T$ where L is a lower triangular matrix. Solve the linear system as follows.

$$LL^T u = y$$

$$L^T u = L^{-1} y = z$$

$$y = Lz$$

Using forward substitution, we can quickly solve for z. Thus leaving

$$L^T u = z$$

which can be solved quickly using backward substitution, thus we have obtained u and therefore v is the unique solution to the Dirichlet problem.

The matrix calculations are made in Java using a Matrix data type that I created. It allows for addition, multiplication, and Choleski factorization, among other things. The solution is calculated fairly quickly due to the ease of calculating the Choleski factorization.

4 Java Source Code

4.1 DrawNetwork.java

```
//This applet implements a Graphical User Interface for manipulation and
//calculations involving electrical networks

import java.awt.*;
import java.lang.Double;
import java.lang.Math;
import java.lang.NumberFormatException;
import Network;
import VoltDialog;
import Matrix;
import LatticeDialog;
import circleDialog;

public class DrawNetwork extends java.applet.Applet {

/*******************************************************************************
Class Variables
*******************************************************************************/
Network N;                 //the electrical network
   Point[] points; //the points to be drawn onscreen
   static final int maxpoints = 300; //maximum number of points
   static final int notset = -1; //used when determining what 2 points
    //user has clicked on
   int moveflag; //if a point is currently being moved
   Choice grp;             //drop down menu
   int a, b; //indices of points we wish to connect
Image offscreenImg;        //used in double buffering
   Graphics offscreenG;
   TextField tf;        //where user inputs conductivity
   Checkbox volt, conduct, numbering;
   boolean toomany; //used to determine if user is trying to incorrectly
    //enter boundary nodes into network


/*******************************************************************************
init() function
*******************************************************************************/
   //initialize the applet
   public void init() {
    N = new Network();  //call Network constructor
      points = new Point[maxpoints]; //create array of points
      moveflag = maxpoints; //won't ever move this point in array

      a = notset;  //we aren't moving any indices
      b = notset;
      toomany = false; //user hasn't entered any boundary nodes yet

//add various user interface elements to screen
      Panel intfc = new Panel();
      Panel choices = new Panel();
      intfc.setLayout (new FlowLayout(FlowLayout.RIGHT));
      choices.setLayout (new FlowLayout(FlowLayout.RIGHT));
      intfc.setBackground(Color.white);
      choices.setBackground(Color.white);
      setLayout(new BorderLayout());
      grp = new Choice();
      grp.addItem("Add Boundary Node");
      grp.addItem("Add Interior Node");
      grp.addItem("Move Node");
      grp.addItem("Remove a Node");
      grp.addItem("Set Conductance");
      grp.addItem("Remove Connection");
      grp.addItem("Enter Boundary Voltages");
      grp.addItem("Construct Lattice");
      grp.addItem("Construct Circular Network");
      tf = new TextField("",5);
      numbering = new Checkbox("Show Node Numbering");
      volt = new Checkbox("Show Voltage");
      conduct = new Checkbox("Show Conductivity");
      choices.add(numbering);
      choices.add(volt);
      choices.add(conduct);
      intfc.add(new Button("Restart"));
      intfc.add(new Button("Dirichlet"));
      intfc.add(new Label("   "));
      intfc.add(grp);
      intfc.add(new Label("Enter Conductance"));
      intfc.add(tf);
      add("North",intfc);
      add("South",choices);
      //create offscreen buffer
      offscreenImg = createImage(size().width,size().height);
      offscreenG = offscreenImg.getGraphics();

      setBackground(Color.white); //set background color of applet
   }

/******************************************************************************
update() function
******************************************************************************/
   //called internally by repaint()
   public void update(Graphics g) {
      paint(g);
   }

/******************************************************************************
paint() function
******************************************************************************/
   //the drawing function
   public void paint(Graphics g) {
    //prepare buffered screen for drawing
      offscreenG.clearRect(0,0,size().width,size().height);

      //draw nodes in color: blue boundary, green interior
      for(int i=0;i<N.numNodes();i++) {
       if(N.onBoundary(i)) {
          offscreenG.setColor(Color.blue);
       offscreenG.fillOval(points[i].x-3,points[i].y-3,6,6);
         }
         else {
          offscreenG.setColor(Color.green);
            offscreenG.fillOval(points[i].x-3,points[i].y-3,6,6);
         }
         //draw node numbering
         if(numbering.getState()) {
          offscreenG.setColor(Color.black);
          offscreenG.drawString("# "+i,points[i].x+5,points[i].y+6);
         }
         //draw voltages
         if(volt.getState() && N.known(i)) {
```

```
         offscreenG.setColor(Color.black);
           offscreenG.drawString("v("+i+")="+N.volts(i),points[i].x+5,
               points[i].y-6);
        }
        //draw connections
        for(int j=0;j<i;j++)
           if(N.connected(i,j)) {
            offscreenG.setColor(Color.black);
              offscreenG.drawLine(points[i].x,points[i].y,
                 points[j].x,points[j].y);
              if(conduct.getState()) {
               int midx, midy; //midpoint coordinates
               midx = (int)(points[i].x*.5+points[j].x*.5+5);
               midy = (int)(points[i].y*.5+points[j].y*.5+5);
               //print conductivities at midpoint
               offscreenG.drawString(Double.toString(N.conductance(i,j)),
                  midx,midy);
              }
           }
     }

     //draw offscreen buffer
     g.drawImage(offscreenImg,0,0,this);
  }


/*******************************************************************************
mouseDown() function
*******************************************************************************/
  //if user clicks see what is going on
  public boolean mouseDown(Event evt, int x, int y) {
   Point point = new Point(x,y);
   if(grp.getSelectedItem().equals("Add Boundary Node")) {
   //add a boundary node to the network if not full and there are no
        //interior nodes yet
        if(N.numNodes() < maxpoints && !toomany) {
   N.addNode(true);
      //add point to array of points
      points[N.numNodes()-1] = new Point(x,y);
         a=b=notset;
         //redraw screen
      repaint();
           return true;
        }
     }
     else if(grp.getSelectedItem().equals("Add Interior Node")) {
      //add an interior node to the network if not full
        if(N.numNodes() < maxpoints) {
         N.addNode(false);
         //add point to array of points
      points[N.numNodes()-1] = new Point(x,y);
         a=b=notset;
           toomany = true; //user can no longer enter boundary nodes
         //redraw screen
      repaint();
           return true;
        }
     }
     //check if user wants to remove a node
     else if(grp.getSelectedItem().equals("Remove a Node")) {
      //find out which node it is
        for(int i=0;i<N.numNodes();i++)
           for(int j=-2;j<3;j++)
              for(int l=-2;l<3;l++)
                 if(point.equals(new Point(points[i].x+j,
                                           points[i].y+l))) {
                    N.removeNode(i);
                    //remove from point listing
                    for(int s=i;s<N.numNodes();s++)
                     points[s] = points[s+1];
                    repaint();
                    return true;
                 }
     }
     //check if user wanted to move a node
     else if(grp.getSelectedItem().equals("Move Node")) {
        a=b=notset;
        //check if user clicked on a node
        for(int i=0;i<N.numNodes();i++)
           for(int j=-2;j<3;j++)
              for(int l=-2;l<3;l++)
                 if(point.equals(new Point(points[i].x+j,
                                           points[i].y+l))) {
                  moveflag = i; //this identifies which point to be moved
```

```
                  return true;
               }
      }
      //check if user wants to connect two nodes
      else if (grp.getSelectedItem().equals("Set Conductance")) {
         for(int i=0;i<N.numNodes();i++)
            for(int j=-2;j<3;j++)
               for(int l=-2;l<3;l++)
                  if(point.equals(new Point(points[i].x+j,
                                            points[i].y+l))) {
                     if(a == notset)
                      a = i;
                     else if(b == notset && a!=i) {
                      try {
                      b = i;
                         //connect two nodes but watch for error
N.connect(a,b,(new Double(tf.getText())).doubleValue());
                         a = b = notset;
                         repaint();
                        }
                        catch(NumberFormatException e) {}
                     }
                     return true;
                   }
      }
      else if(grp.getSelectedItem().equals("Remove Connection")) {
       for(int i=0;i<N.numNodes();i++)
            for(int j=-2;j<3;j++)
               for(int l=-2;l<3;l++)
                  if(point.equals(new Point(points[i].x+j,
                                            points[i].y+l))) {
                   if(a == notset)
                      a = i;
                     else if(b == notset && a!=i) {
                      b = i;
                        //unconnect two nodes
N.unConnect(a,b);
                        a = b = notset;
                        repaint();
                     }
                     return true;
                   }
      }
      return false;
   }


/***********************************************************************
mouseDrag() function
***********************************************************************/
   //check if user is dragging a point
   public boolean mouseDrag(Event evt, int x, int y) {
      //check if user is trying to drag an old point
      if(moveflag < N.numNodes() && grp.getSelectedItem().equals("Move Node")) {
         points[moveflag].move(x,y); //move the point
         repaint();
      }
      return true;
   }


/***********************************************************************
mouseUp() function
***********************************************************************/
   //after user unclicks, don't move anymore
   public boolean mouseUp(Event evt, int x, int y) {
      moveflag = N.numNodes();
      return true;
   }


/***********************************************************************
action() function
***********************************************************************/
   //watch for events
   public boolean action(Event evt, Object arg) {
    //check if user is going to set the conductance
    if(arg.equals("Set Conductance")) {
       //default to one
tf.setText("1");
         tf.selectAll();
      }
      //does user want to enter the boundary voltages?
```

```
else if(arg.equals("Enter Boundary Voltages")) {
 //create dialog box
 VoltDialog b_volts = new VoltDialog(N,N.numBoundary(),this);
   b_volts.move(this.location().x+200,this.location().y+200);
   b_volts.resize(250,150);
   b_volts.setTitle("Boundary Voltages");
 b_volts.show();
}
else if(arg.equals("Construct Lattice")) {
 //create dialog box
 LatticeDialog l = new LatticeDialog(this);
   l.move(this.location().x+200,this.location().y+200);
   l.resize(250,200);
   l.setTitle("Construct Lattice");
   l.show();
}
else if(arg.equals("Construct Circular Network")) {
 //create dialog box
 circleDialog c = new circleDialog(this);
   c.move(this.location().x+200,this.location().y+200);
   c.resize(350,200);
   c.setTitle("Construct Circular Network");
   c.show();
}
//check if we are going to solve Dirichlet Problem
else if(arg.equals("Dirichlet")) {
 //see if the boundary voltages have been entered
   boolean b = true;
   for(int i=0;i<N.numBoundary();i++)
    if(!N.known(i))
       b=false;
   if(b)
 solve();
}
else if(arg.equals("Add Boundary Node"))
 tf.setText("");
else if(arg.equals("Add Interior Node"))
 tf.setText("");
else if(arg.equals("Remove a Node"))
 tf.setText("");
else if(arg.equals("Move Node"))
 tf.setText("");
```

```
      else if(evt.target.equals(numbering) || evt.target.equals(volt) ||
         evt.target.equals(conduct))
       repaint();
      else if(arg.equals("Restart")) {
       //re-initialize everything
       N = new Network();
         points = new Point[maxpoints];
       moveflag = maxpoints;
         a=b=notset;
         toomany = false;
repaint();
      }
else return super.action(evt,arg);
      return true;
   }


/*******************************************************************************
solve() function
*******************************************************************************/
//this function determines interior voltages of network given boundary
   //voltages
   public void solve() {
    int n = N.numInterior();
      int m = N.numBoundary();
System.out.println(N.toString());
      //Create C
      Matrix C = new Matrix(n,n);
      for(int i=m;i<N.numNodes();i++)
       for(int j=m;j<N.numNodes();j++) {
          if(i == j) {
               //check all nodes if they are neighbors
             for(int l=0;l<N.numNodes();l++)
                  if(N.connected(i,l)) {
                   C.set(i-m,j-m,C.get(i-m,j-m)+N.conductance(i,l));
                  }
            }
            else {
             //if nodes representing row i and column j are connected, then
               //their entry in K is the negative of the conductivity between
               //them, otherwise it is zero
             if(N.connected(i,j))
          C.set(i-m,j-m,-N.conductance(i,j));
      }
    }
  //create b
  Matrix Bt = new Matrix(n,m);
  for(int i=m;i<N.numNodes();i++)
   for(int j=0;j<m;j++) {
      if(i == j) {
           //check all nodes if they are neighbors
         for(int l=0;l<N.numNodes();l++)
               if(N.connected(i,l)) {
                Bt.set(i-m,j,Bt.get(i-m,j)+N.conductance(i,l));
               }
        }
        else {
         //if nodes representing row i and column j are connected, then
           //their entry in K is the negative of the conductivity between
           //them, otherwise it is zero
         if(N.connected(i,j))
            Bt.set(i-m,j,-N.conductance(i,j));
        }
    }
  Matrix phi = new Matrix(N.numBoundary(),1);
  for(int i=0;i<N.numBoundary();i++)
   phi.set(i,0,-N.volts(i));
  Matrix b = new Matrix(Bt.mult(phi));

//since C is symmetric, positive definite, we can factor C into LLt
  //by Choleski factorization
  Matrix L = new Matrix(C.choleski());
  //solve LLt*x=b
  double sum;
  Matrix y = new Matrix(n,1);
  Matrix x = new Matrix(n,1); //this will be the solution
y.set(0,0,b.get(0,0)/L.get(0,0));
  for(int i=1;i<n;i++) {
   sum = 0;
     for(int j=0;j<i;j++)
      sum += L.get(i,j)*y.get(j,0);
     y.set(i,0,(b.get(i,0)-sum)/L.get(i,i));
  }
  x.set(n-1,0,y.get(n-1,0)/L.get(n-1,n-1));
```

```
      for(int i=n-2;i>=0;i--) {
       sum = 0;
         for(int j=i+1;j<n;j++)
          sum += L.get(j,i)*x.get(j,0);
         x.set(i,0,(y.get(i,0)-sum)/L.get(i,i));
      }
      //x is solution to Dirichlet problem, put voltages in network
      for(int i=N.numBoundary();i<N.numNodes();i++)
       N.setVolts(i,x.get(i-N.numBoundary(),0));
      repaint();
   }


/*******************************************************************************
lattice() function
*******************************************************************************/
//this function constructs a rectangular lattice given the number of rows
   //and columns
   public void lattice(int rows, int cols) {
    int numbound = 2*rows + 2*cols; //the number of boundary nodes
      int numinterior = rows*cols;
      int numtotal = numbound + numinterior;
      int w = (int)((size().width-100)/(cols+1)); //width between each node
      int h = (int)((size().height-110)/(rows+1)); //height between each node

      //check if user is trying to enter too many nodes
      if(numtotal > maxpoints)
       return;

    //re-initialize all data
      N = new Network();
      points = new Point[maxpoints];
      moveflag = maxpoints;
      a=b=notset;
      toomany = false;

      //enter boundary nodes into Network data structure
      for(int i=0;i<numbound;i++) {
       N.addNode(true);
         //determine position of node
         if(i<cols)  //top of lattice
          points[i] = new Point(50+w+w*i,60);
         else if(i>=cols && i<(cols+rows)) //right side of lattice
          points[i] = new Point(size().width-50,60+h+h*(i-cols));
         else if(i>=(rows+cols) && i<(2*cols+rows)) //bottom of lattice
          points[i] = new Point(50+w*(2*cols+rows-i),60+h+rows*h);
         else //left side of lattice
          points[i] = new Point(50,60+h*(numbound-i));
      }
      toomany = true; //can't enter any more boundary nodes
      //enter interior nodes into Network
      int adj = -1;
      for(int i=0;i<numinterior;i++) {
       N.addNode(false);
         //determine position of node
         if(i%cols == 0)
          adj++;
         points[i+numbound] = new Point(50+w+w*(i%cols),60+h+h*adj);
      }
      //make connections with conductivity of 1 for boundary nodes
      for(int i=0;i<numbound;i++) {
       //determine position of node
         if(i<cols)  //top of lattice
          N.connect(i,i+numbound,1);
         else if(i>=cols && i<(cols+rows)) //right side of lattice
          N.connect(i,numbound+(cols-1)+cols*(i-cols),1);
         else if(i>=(rows+cols) && i<(2*cols+rows)) //bottom of lattice
          N.connect(i,numtotal-1+(rows+cols-i),1);
         else //left side of lattice
            N.connect(i,numbound+cols*(numbound-i-1),1);
      }
      //make connections for interior nodes
      for(int i=numbound;i<numtotal;i++) {
       //check if corner node
         if(i==numbound) { //upper left corner
          N.connect(i,i+1,1);
            N.connect(i,i+cols,1);
         }
         else if(i==(numbound+cols-1)) { //upper right corner
          N.connect(i,i-1,1);
            N.connect(i,i+cols,1);
         }
         else if(i==(numtotal-1)) { //lower right corner
          N.connect(i,i-1,1);
```

```
          N.connect(i,i-cols,1);
        }
        else if(i==(numtotal-cols)) { //lower left corner
         N.connect(i,i+1,1);
          N.connect(i,i-cols,1);
        }
        //check if on an outer edge
        else if(i<(numbound+cols-1) && i>numbound) { //top edge
         N.connect(i,i-1,1);
          N.connect(i,i+1,1);
          N.connect(i,i+cols,1);
        }
        else if(i<(numtotal-1) && i>(numtotal-cols+1)) { //bottom edge
         N.connect(i,i-1,1);
          N.connect(i,i+1,1);
          N.connect(i,i-cols,1);
        }
        else if((i-numbound)%cols == 0) { //left edge
         N.connect(i,i+1,1);
          N.connect(i,i+cols,1);
          N.connect(i,i-cols,1);
        }
        else if((i+1-numbound)%cols == 0) { //right edge
         N.connect(i,i+cols,1);
          N.connect(i,i-cols,1);
          N.connect(i,i-1,1);
        }
        else { //do the rest
         N.connect(i,i-1,1);
          N.connect(i,i+1,1);
          N.connect(i,i+cols,1);
          N.connect(i,i-cols,1);
        }
      }
      repaint();
   }

/*******************************************************************************
circular() function
*******************************************************************************/
//this function constructs a circular network given the number of boundary
   //nodes and the radial levels of the graph
```

```
public void circular(int bnd, int r) {
 int maxradius = (int)(Math.min(size().width,size().height)/2.0)-50;
   Point center = new Point((int)(size().width/2),(int)(size().height/2));
   int numinterior = (r-1)*bnd+1;
   int numtotal = numinterior + bnd;
   double shift = Math.PI/4;

   //check if user wants too many nodes
   if(numtotal > maxpoints)
    return;

   //re-initialize all data
   N = new Network();
   points = new Point[maxpoints];
   moveflag = maxpoints;
   a=b=notset;
   toomany = false;
   //add boundary nodes to network
   for(int i=0;i<bnd;i++) {
    N.addNode(true);
      points[i] = new Point((int)(center.x+maxradius*
        Math.cos(2*Math.PI*i/bnd+shift)),
        (int)(center.y+maxradius*
                              Math.sin(2*Math.PI*i/bnd+shift)));
   }
   //add interior nodes to network
   for(int i=bnd;i<numtotal;i++) {
    N.addNode(false);
      points[i]=new Point((int)(center.x+maxradius*(r-Math.floor(i/bnd))/r*
         Math.cos(2*Math.PI*i/bnd+shift)),
         (int)(center.y+maxradius*(r-Math.floor(i/bnd))/r*
                               Math.sin(2*Math.PI*i/bnd+shift)));
   }
   //make connections
   for(int i=0;i<(r-1)*bnd;i++) {
    N.connect(i,i+bnd,1);
   }
   for(int i=(r-1)*bnd;i<numtotal-1;i++) {
    N.connect(i,numtotal-1,1);
   }
   for(int i=bnd;i<numtotal-1;i++) {
      if(i%bnd == bnd-1)
```

```
          N.connect(i,i-bnd+1,1);
         else
       N.connect(i,i+1,1);
      }
      //redraw screen
      repaint();
   }

/*******************************************************************************
handleEvent() function
*******************************************************************************/
//this function deals with the destruction of the applet
public boolean handleEvent(Event evt) {
if (evt.id == Event.WINDOW_DESTROY)
    System.exit(0);
    return super.handleEvent(evt);
}

}//end of class DrawNetwork
```

4.2 Network.java

```
//This class implements an Electrical Network data structure
import Node;
import java.lang.Double;

public class Network {
   private static final int maxverts = 300;
   private int V; //number of nodes
   private Node[] vertices; //holds all of the nodes in the network
   private double[][] edges; //adjacency matrix with conductances, if
    //conductivity is NaN, then there is no
                              //connection

//converts Network class to a string which can be output
   public String toString() {
    String s = new String("Number of Nodes = "+String.valueOf(V)+"\n");
      //output each node info
      s=s.concat("Node Information:\n");
      for(int i=0;i<V;i++)
       s=s.concat(String.valueOf(i) + " " + vertices[i].toString() + "\n");
      //output adjacency matrix
      s=s.concat("Adjacency Matrix:\n   [ ");
      for(int i=0;i<V;i++) {
         for(int j=0;j<V;j++) {
            s=s.concat(String.valueOf(edges[i][j]));
            s=s.concat(" ");
         }
         if(i!=V-1)
            s=s.concat("\n     ");
      }
      s=s.concat("]\n");

return s;
   }

   //Constructor creates Network with no nodes, they must be added later
   public Network() {
    V = 0; //there are no nodes
      vertices = new Node[maxverts];
      edges = new double[maxverts][maxverts];
   }

   //returns the voltage at node i
   public double volts(int i) {
    return vertices[i].voltage;
   }

   //sets the voltage at node i
   public void setVolts(int i, double v) {
    vertices[i].voltage = v;
      vertices[i].known = true;
      return;
   }

   //returns true if node i is on boundary, false otherwise
   public boolean onBoundary(int i) {
    return vertices[i].boundary;
   }

   //sets a node to be boundary node or not
   public void setBoundary(int i, boolean b) {
    vertices[i].boundary = b;
      return;
   }

   //returns true if voltage at node i is known, false otherwise
   public boolean known(int i) {
    return vertices[i].known;
   }

   //sets the voltage to be known or not at node i
   public void setKnown(int i, boolean b) {
    vertices[i].known = b;
      return;
   }

   //returns true if node i and j are connected, false otherwise
   public boolean connected(int i, int j) {
    if(Double.isNaN(edges[i][j]))
       return false;
      else
       return true;
   }

   //connects two nodes by a conductor in adjacency matrix
   public void connect(int i, int j, double conductivity) {
edges[i][j] = conductivity;
      edges[j][i] = conductivity;
      return;
   }

   //removes a connection between nodes i and j
   public void unConnect(int i, int j) {
    edges[i][j] = edges[j][i] = Double.NaN;
   }

   //adds a node to the network with boundary information given by b
   public void addNode(boolean b) {
    V++;
      vertices[V-1] = new Node(b);
      for(int i=0;i<V;i++) { //node is not connected to anything
       edges[V-1][i] = Double.NaN;
         edges[i][V-1] = Double.NaN;
      }
      return;
```

```
}

//removes a nodes from the network
public void removeNode(int n) {
 //collapse voltage vector
   for(int i=n;i<V-1;i++)
    vertices[i] = vertices[i+1];
   //collapse conductivity matrix, column first
   for(int i=0;i<V;i++)
    for(int j=n;j<V-1;j++)
       edges[i][j] = edges[i][j+1];
   //collapse the row
   for(int i=n;i<V-1;i++)
    for(int j=0;j<V-1;j++)
       edges[i][j] = edges[i+1][j];
   V--;
}

//returns the current flowing between two nodes using Ohm's Law
public double current(int i, int j) {
 return edges[i][j]*(vertices[j].voltage-vertices[i].voltage);
}

//returns the conductance between two nodes
public double conductance(int i, int j) {
 return edges[i][j];
}

//returns the number of nodes in the network
public int numNodes() {
 return V;
}

//returns the number of boundary nodes
public int numBoundary() {
   for(int i=0;i<V;i++)
    if(!vertices[i].boundary)
       return i;
   return V;
}

//returns the number of interior nodes
```

```
   public int numInterior() {
      for(int i=0;i<V;i++)
       if(!vertices[i].boundary)
          return V-i;
      return 0;
   }

   //returns the number of neighbors connected to node i
   public int neighbors(int i) {
    int counter = 0;
      for(int j=0;j<V;j++)
       if(connected(i,j))
          counter++;
return counter;
   }


}
```

4.3 Node.java

```
//A node in a network

public class Node {
public double voltage;
   public boolean known; //is the voltage of this node known or not
   public boolean boundary; //is this node on boundary or not

   //Constructors
   public Node(double v, boolean b) {
    voltage = v;
      boundary = b;
      known = true;
   }
   //if the voltage is not known
   public Node(boolean b) {
    boundary = b;
      known = false;
   }
   public Node(Node n) {
    boundary = n.boundary;
```

```
      known = n.known;
      if(known)
       voltage = n.voltage;
   }
//output a Node as a string
   public String toString() {
    String s = new String();
    if(known)
s=s.concat("Voltage = "+String.valueOf(voltage)+"\n");
      if(boundary)
       s=s.concat("Boundary Node\n");
      else
       s=s.concat("Interior Node\n");
      return s;
   }
}
```

4.4 Matrix.java

```
//Matrix class with no error checking

import java.lang.Math;

public class Matrix {
   private int cols;          //number of columns
   private int rows;          //number of rows
   private double[][] mat;    //the 2-D matrix

   //Constructor
   public Matrix(int r, int c) {
      rows = r;
      cols = c;
      mat = new double[rows][cols];
      //initialize to zeros
      for(int i=0;i<rows;i++)
         for(int j=0;j<cols;j++)
            mat[i][j] = 0;
   }

   //Construct a new Matrix from an old one
   public Matrix(Matrix m) {
      rows = m.rowsize();
      cols = m.colsize();
      mat = new double[rows][cols];
      for(int i=0;i<rows;i++)
         for(int j=0;j<cols;j++)
            mat[i][j] = m.get(i,j);
   }

   //this method sets the value of the matrix at r,c to v
   public void set(int r, int c, double v) {
      mat[r][c] = v;
   }

   //this method gets the value of the matrix at r,c
   public double get(int r, int c) {
      return mat[r][c];
   }

   //returns the number of columns
   public int colsize() {
      return cols;
   }

   //returns the number of rows
   public int rowsize() {
      return rows;
   }

   //multiplies this matrix by another one
   public Matrix mult(Matrix m) {
      Matrix ans = new Matrix(rows,m.colsize());
      double c;
      for(int i=0;i<ans.rowsize();i++)
         for(int j=0;j<ans.colsize();j++) {
            c = 0;
            for(int k=0;k<m.rowsize();k++)
               c += mat[i][k]*m.get(k,j);
            ans.set(i,j,c);
         }
      return ans;
   }

   //adds this matrix to another one
```

```
public Matrix add(Matrix m) {
   Matrix ans = new Matrix(rows,cols);
   for(int i=0;i<rows;i++)
      for(int j=0;j<cols;j++)
         ans.set(i,j,mat[i][j]+m.get(i,j));
   return ans;
}

//converts this matrix to a string which can then be
//printed out
public String toString() {
   String s = new String("[");
   for(int i=0;i<rows;i++) {
      for(int j=0;j<cols;j++) {
         s=s.concat(String.valueOf(mat[i][j]));
         s=s.concat(" ");
      }
      if(i!=rows-1)
         s=s.concat(String.valueOf('\n'));
   }
   s=s+"]";
   return s;
}

//this method subtracts two matrices
public Matrix sub(Matrix m) {
   Matrix ans = new Matrix(rows,cols);
   for(int i=0;i<rows;i++)
      for(int j=0;j<cols;j++)
         ans.set(i,j,this.get(i,j)-m.get(i,j));
   return ans;
}

//this method returns the transpose of the current matrix
public Matrix trans() {
   Matrix m = new Matrix(cols,rows);
   for(int i=0;i<cols;i++)
      for(int j=0;j<rows;j++)
         m.set(i,j,mat[j][i]);
   return m;
}
```

```
   //factor a symmetric, positive definite matrix into LLt by Choleski
   //factorization assuming it is a positive definite, symmetric, square matrix
   public Matrix choleski() {
    double sum;
      int n = rows;
      Matrix L = new Matrix(n,n);  //the lower triangular matrix

      L.set(0,0,Math.sqrt(mat[0][0]));
      for(int j=1;j<n;j++)
       L.set(j,0,mat[j][0]/L.get(0,0));
      for(int i=1;i<n-1;i++) {
       sum = 0;
         for(int k=0;k<i;k++)
          sum += L.get(i,k)*L.get(i,k);
         L.set(i,i,Math.sqrt(mat[i][i]-sum));
         for(int j=i+1;j<n;j++) {
          sum = 0;
            for(int k=0;k<i;k++)
             sum += L.get(j,k)*L.get(i,k);
            L.set(j,i,1/L.get(i,i)*(mat[j][i]-sum));
         }
      }
      sum = 0;
      for(int k=0;k<n-1;k++)
       sum += L.get(n-1,k)*L.get(n-1,k);
      L.set(n-1,n-1,Math.sqrt(mat[n-1][n-1]-sum));
      return L;
   }
}
```

4.5 VoltDialog.java

```
//this class provides a dialog box for the DrawNetwork class which prompts
//the user to enter the boundary voltages of the network

import java.awt.*;
import java.lang.Double;
import Network;
import java.applet.Applet;

class VoltDialog extends Frame {
```

```
/*****************************************************************************
Class Variables
*****************************************************************************/
TextField text; //where user enters boundary voltages
   int node = 0; //the node number for which we are setting the voltage
   double[] allvolts;   //the stored boundary voltages
   int boundary; //number of boundary nodes
   Network daddy; //the applet which is creating this dialog box
   Applet a;


/*****************************************************************************
VoltDialog() Constructor
*****************************************************************************/
//constructor
public VoltDialog(Network N, int V, Applet b) {
    a = b;
    daddy = new Network();
      daddy = N;
      boundary = V;
      allvolts = new double[V];
      Panel p1 = new Panel();
      p1.add(new Label("Enter Boundary Voltage: "));
      p1.add(text = new TextField("",10));
      add("Center",p1);
      Panel p2 = new Panel();
      p2.add(new Button("Cancel"));
      add("South",p2);
      text.setText("Node 0");
      text.selectAll();
   }


/*****************************************************************************
action() function
*****************************************************************************/
   //handles the events that occur
   public boolean action(Event evt, Object arg) {
    if(arg.equals("Cancel"))
       dispose();
      //check if user has entered a voltage
      else if(evt.target == text) {
       if(node < boundary-1) {
       Double voltage = new Double(text.getText()); //turn text into number
          //put voltage in array
          allvolts[node++] = voltage.doubleValue();
         //display the node number, prompting user for next entry
          text.setText("Node "+node);
          text.selectAll();
         }
         //we're done
       else {
            Double voltage = new Double(text.getText()); //turn text into number
          //put voltage in array
          allvolts[node++] = voltage.doubleValue();
       //set the new boundary voltages within the current network
          for(int i=0;i<daddy.numBoundary();i++)
          daddy.setVolts(i,allvolts[i]);
          //make interior voltages not known
          for(int i=daddy.numBoundary();i<daddy.numNodes();i++)
            daddy.setKnown(i,false);
       dispose(); //remove resources taken by pop-up box
            a.repaint();
    }
      }
      else
       return super.action(evt, arg);
      return true;
   }


/******************************************************************************
handleEvent() function
******************************************************************************/
   //if user closes window by hitting exit button, handle it
   public boolean handleEvent(Event evt) {
    if (evt.id == Event.WINDOW_DESTROY)
       dispose();
      else
       return super.handleEvent(evt);
      return true;
   }
}
```

4.6 circleDialog.java

```
//this class implements a pop up box that asks the user for the number of
//boundary nodes

import java.awt.*;
import java.lang.Integer;
import DrawNetwork;

public class circleDialog extends Frame {
TextField numbound, numlevels;
   int numboundary, radial_levels;
   DrawNetwork a;

   //constructor
circleDialog(DrawNetwork d) {
    a = new DrawNetwork();
      a = d;
    Panel p1 = new Panel();
      p1.add(new Label("Enter Number of Boundary Nodes "));
      p1.add(numbound = new TextField("",5));
      Panel p2 = new Panel();
      p2.add(new Label("Enter Number of Radial Levels "));
      p2.add(numlevels = new TextField("",5));
      Panel p3 = new Panel();
      p3.add(new Button("OK"));
      add("North",p1);
      add("Center",p2);
      add("South",p3);
   }

   public boolean action(Event evt, Object arg) {
      //check if we're done
      if(arg.equals("OK")) {
         numboundary = (new Integer(numbound.getText())).intValue();
         radial_levels = (new Integer(numlevels.getText())).intValue();
         //construct the network with the proper number of boundary nodes
         a.circular(numboundary,radial_levels);
         dispose();
      }
      else
         return super.action(evt,arg);
      return true;
   }

   public boolean handleEvent(Event evt) {
      if(evt.id == Event.WINDOW_DESTROY)
         dispose();
      else
         return super.handleEvent(evt);
      return true;
   }
}
```

4.7 LatticeDialog.java

```
//this class implements a dialog box for creating a lattice network

import java.awt.*;
import DrawNetwork;
import java.lang.Integer;
import java.lang.NumberFormatException;

class LatticeDialog extends Frame {
   TextField rows, cols; //where user enters number of rows & columns
   DrawNetwork a;
   int numrows, numcols;

   //constructor
   public LatticeDialog(DrawNetwork command) {
    a = new DrawNetwork();
      a = command;
      Panel prows = new Panel();
      prows.add(new Label("Enter number of rows: "));
      prows.add(rows = new TextField("",4));
      Panel pcols = new Panel();
      pcols.add(new Label("Enter number of columns: "));
      pcols.add(cols = new TextField("",4));
      Panel p = new Panel();
      p.add(new Button("OK"));
      add("North",prows);
      add("Center",pcols);
      add("South",p);
   }
   public boolean action(Event evt, Object arg) {
      //check if we're done
      if(arg.equals("OK")) {
       //catch an invalid input
         try {
          //get info from textfields an call lattice function
         numrows = (new Integer(rows.getText())).intValue();
          numcols = (new Integer(cols.getText())).intValue();
         a.lattice(numrows,numcols);
          dispose();
         }
         catch(NumberFormatException e) {}
      }
      else
         return super.action(evt,arg);
      return true;
   }

   public boolean handleEvent(Event evt) {
      if(evt.id == Event.WINDOW_DESTROY)
         dispose();
      else
         return super.handleEvent(evt);
      return true;
   }
}
```

4.8 DrawNetwork.html

```
<HEAD>
<TITLE>Electrical Networks</TITLE>
</HEAD>
<BODY>
<H1> Electrical Networks</H1>

<APPLET CODE=DrawNetwork.class WIDTH=800 HEIGHT=500>
</APPLET>
<H2>Instructions</H2>
This applet allows you, the user, the opportunity to interact with Electrical
Networks in a graphical manner.
<P>To create your network, start off by choosing "Add Boundary Node" from
```

```html
the drop-down menu and then click anywhere on the screen, creating boundary
nodes.  The order with which you enter nodes is very important.  Always enter
boundary nodes first in some sort of circular order.  Then choose "Add Interior
Node" from the drop-down menu and place them where you like.  Boundary
nodes are blue and interior nodes are green.  To make a
connection, choose "Set Conductance" from the menu, enter the desired
conductance in the box in the upper right hand corner, and then click on 2
separate nodes.  Continue entering conductances in the box and clicking on 2
nodes until you have made all desired connections.  You can display
information such as node ordering and
the conductivites of the current network by clicking on the appropriate
checkbox.  Press the button "Restart" if you wish to start all over again.  You
can construct a rectangular lattice network by choosing the appropriate choice
from the drop-down menu and then entering the number of rows and columns you
desire. A circular network can be constructed by choosing the proper choice from
the drop down menu and then entering the appropriate number of boundary nodes
and radial levels.  Adjustments to your network can be made as well.  If you would
like to remove a node, remove a connection, or move a node simply choose the
appropriate option from the drop down menu and perform the operation.  If you
attempt to enter more than 299 nodes into the network, the applet won't let you.
After all, who needs more than 299 nodes!?!
<P><H3>Dirichlet Problem</H3>
<P>The Dirichlet problem is as follows:  given boundary voltages of an electrical
network, we wish to find the interior voltages.  The solution exists and is unique.
Start off by allowing node numbering by clicking the "Show Node Ordering" box at the
bottom of the applet.
Now choose "Enter Boundary Voltages" from the drop-down menu.
A pop-up box will come up.  Highlight the text entry box with your mouse.
It should say "Node 0".  Now, enter the appropriate voltages for each node and
press
return, noticing the node number in the box that correlates to the node in your
network.  After all have been entered, allow for the display of node voltages by
clicking on the appropriate box on the bottom of the screen.  Now press the
"Dirichlet" button.
The interior voltages that were calculated by solving a linear
system of equations and the boundary voltages that you entered will be displayed.
You can modify conductivities and boundary
voltages of your current network by choosing the appropriate drop-down menu item.
Then just click "Dirichlet" again and you will see the changes.  Cool, huh!

<P>If you have comments/questions/ideas please e-mail me:
<A HREF="mailto:hoefman@ucla.edu">Mark Hoefer</A><br>
```

```
</BODY>
</HTML>
```

References

[1] Richard L. Burden and J. Douglas Faires, *Numerical Analysis*, 5th ed., PWS Publishing Co., 1993, pp 376-377.

[2] Gary Cornell and Cay S. Horstman, *Core Java*, SunSoft Press, 1996.

[3] E. B. Curtis, D. Ingerman, J. A. Morrow, *Circular Planar Graphs and Resistor Networks*, submitted.