A Computer Algorithm for Finding the Dual of a Polar Network.

Sreekar M. Shastry

June 19, 2003

1 Introduction

In this paper I will discuss the problem of tiling a rectangle by squares and provide an algorithm for creating a tiling given a certain type of graph called a *polar network*. This algorithm will use the facilities provided by Mark Hoefer's DrawNetwork package and will be implemented in the Java programming language.

The algorithm works by taking the incidence relations and circular ordering of edges emanating from each vertex in the given polar network and producing its *polar dual*. Once the Kirchoff matrix of the dual is found, the Dirichlet problem can be solved on both the given network and the dual to produce the tiling. See [3] for details.

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.

2.1 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 [2] that there indeed is a solution and it is unique. This problem can be solved directly by a system of equations.

2.2 The Solution of the Dirichlet Problem

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 [2], 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.

$$LL^Tu = y$$

$$L^Tu = L^{-1}y = z$$

$$y = Lz$$

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

$$L^Tu = z$$

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

3 Tiling Rectangles by Squares

3.1 Introduction

A dissection of a rectangle R into a finite number n of non–overlapping squares is called a *squaring* of R of *order* n. The n squares are the elements of the dissection. "Element" also denotes the length of the sides of the elements.

The problem of characterizing how the elements fit together is overcome by associating a graph (called the "polar net") with each squared rectangle. The placement of the elements of the rectangle are found to be determined by the flow of electric current through this network.

3.2 The Network Associated with a Squared Rectangle

Given a squared rectangle R, consider the point–set formed by the horizontal sides of the elements of R. Its connected components will be horizontal line–segments (each consisting of a set of horizontal sides of elements of R); enumerate them as $l_1, l_2, \ldots, l_n$, where l_1, l_n are the upper and lower edges of R. Place a point v_i on the midpoint of l_i. For each element E in R, its upper edge will lie in

some l_i and its lower edge in some l_j ($i \neq j$). Join the points v_i, v_j by an edge e. The points v_1, v_n are the *poles* of the network. Let the north pole be $v_N = v_1$ and the south pole be $v_S = v_n$. This graph is the polar network (p–net) of the rectangle R. It is clear that every p–net is planar and that no circuit encloses a pole.

p–nets cannot contain loops, multiple edges connecting two vertices, or vertices with degree 1 or 2.

3.3 Construction of the Dual Network

Let $\mathcal{P}$ be the p–net associated with a rectangle R. The dual network $\mathcal{P}'$ is constructed as follows. For each edge $e \in \mathcal{P}$, draw the dual edge e' as the perpendicular bisector of e. In each face f_i in $\mathcal{P}$, place a vertex u_i. Connect every edge e' which has an end in f_i to u_i. Draw a circular boundary around $\mathcal{P}$ which contains its poles. Even though the boundary does not consist of edges, consider the regions defined by the boundary and the edges from pole to pole to be faces. Place two more vertices in these faces and join the remaining unconnected endpoints of edges to these vertices. These vertices are the poles of the dual network. See Fig. 1 for an example of a p–net and its dual.

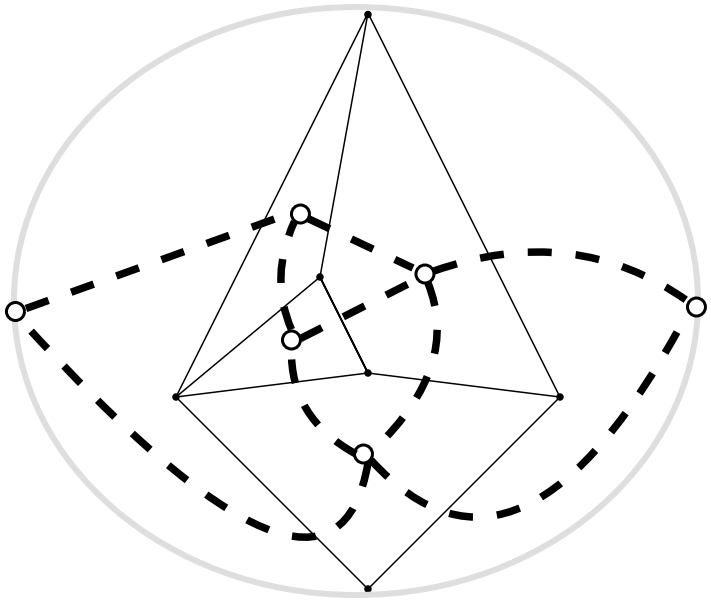

Fig. 1. $\mathcal{P}$ with $\mathcal{P}'$ in dashed lines.

Let $\mathcal{P} = (\{v_i\}, \{e_j\})$ and $\mathcal{P}' = (\{u_k\}, \{e'_j\})$. If $\mathcal{P}$ has F faces then it is clear that $|\{e'_j\}| = |\{e_j\}|$ and $|\{v_i\}| = F + 2$ is not necessarily equal to $|\{u_k\}|$.

Let $e = v_r v_s \in \mathcal{P}$, and have potentials V_r, V_s, and conductivity γ_{rs}, then the current flowing across e is $I_{rs} = \gamma_{rs}(V_r - V_s)$. Let the potentials at the endpoints of e' be I_r and I_s such that $I_{rs} = I_r - I_s$. The conductivity of e' will then be $1/\gamma_{rs}$.

Let C be a first cofactor of the Kirchhoff Matrix for $\mathcal{P}$. It is known that all first cofactors are equal and that the total current flowing in $\mathcal{P}$ is given by this first cofactor. Let the potential drop between the poles of $\mathcal{P}$ be V and the total current and potential drop in $\mathcal{P}'$ be C' and V' respectively. Then $C = V'$ and $C' = V$.

If all of the conductivities are taken to be unity, then C is also the number of spanning trees in $\mathcal{P}$. This is the well known Matrix–Tree Theorem.

3.4 The Correspondence between p–*nets* and Squared Rectangles

In a p–net for which the only sources and sinks of current are at the poles, it is easy to see that the potential of each interior vertex lies between that of the poles. This fact implies that for every interior vertex, all of the incoming currents are contained in an angle (in the plane) and all outgoing currents are contained in the reflex of that angle. Suppose that at interior vertex v_i, two outgoing currents separate (in the plane) two ingoing ones. By Kirchhoff's Law, since there are outgoing currents there is a vertex at a lower potential, and similarly, since there are incoming currents, there is a vertex at higher potential. Since potential reaches its maximum and minimum values at the poles, each of the chains of higher and lower potential emanating from v_i can be continued to the poles. As the graph is planar, this means that a chain of increasing potential crosses a chain of decreasing potential, i.e. that there exists a v_j $(i \neq j)$ such that its potential is both higher and lower than the potential at v_i. This is a contradiction, which proves that there do not exist saddle points of current at a vertex in a p–net.

To show that p–nets correspond to squared rectangles, we first take all conductances to be one. Suppose an edge $e \in \mathcal{P}$ has its endpoints at potentials V_1, V_2 and its dual e' has its endpoints at potentials V_1', V_2'. If $V_1 < \mu < V_2$ then e *comprises* $(\ , \mu)$. If $V_1' < \lambda < V_2'$ then e *comprises* $(\lambda,\ )$. If both relations hold, then e *comprises* (λ, μ).

Since the conductance is unity, $V_2 - V_1 =$ current in $e =$ current in $e' = V_2' - V_1'$. In a rectangle R of height V and base C, for each edge e in $\mathcal{P}$ draw a square whose horizontal sides are at a height V_1, V_2 above the base and whose vertical sides are at a distance V_1', V_2' to the right of the left–hand vertical side. In [1] it is shown that the non–existence of saddle points of current implies that for any given λ which is not equal to the potential of any node in $\mathcal{P}$, there is a unique chain of wires in $\mathcal{P}$ comprising $(\lambda,\ )$. Similarly, there is a unique chain of wires in $\mathcal{P}$ comprising $(\mu,\ )$. Thus for every (λ, μ) such that $0 < \lambda < C$, and $0 < \mu < V$, with $\lambda \neq$ the potential of any vertex in $\mathcal{P}'$ and $\mu \neq$ the potential of any vertex in $\mathcal{P}$, there is exactly one edge comprising (λ, μ). Thus, the rectangle R is tiled completely and without overlap (except at the boundaries of the squares).

The intuitive reason that p–nets correspond to tiled rectangles is that the top and bottom sides of the rectangle can be considered as sources of length and in the interior of the rectangle, length is conserved, just as current would be in a network with two poles.

4 The Algorithm

Each vertex v_i of $\mathcal{P}$ has associated with it a circular ordering of edges $C_i = (e_1, e_2, \ldots, e_{\deg v_i})$. The successor of an edge in C_i is defined as

$$\mathrm{succ}(e_j) = \begin{cases} e_{j+1}, & \text{if } 1 \le j < \deg v_i \\ e_1, & \text{if } j = \deg v_i \end{cases} .$$

C_i is obtained by looking at the rectangular coordinates of the vertices $\{v_j\}$ adjacent to v_i. Graphically, a vertex v_j is represented in the computer by a pair of rectangular coordinates (x_j, y_j), relative to some fixed origin. First, write down a set of pairs $\Theta = \{(v_j, \theta_j)\}$ such that

$$\theta_j = \arctan\left(\frac{y_j - y_i}{x_j - x_i}\right)$$

where $v_i = (x_i, y_i)$ is the vertex about which we want to find the circular ordering. Next sort Θ on the θ_j and read off C_i. The ordering is arbitrary and is chosen to be counterclockwise.

If $\mathcal{P} = (\{v_i\}, \{e_j\})$ and $\mathcal{P}' = (\{u_k\}, \{e'_j\})$, then let the jth edge of both $\mathcal{P}$ and $\mathcal{P}'$ be represented in the program by a quadruple of integers $X_j = (p, q, p', q')$ such that $e = v_p v_q$ and $e' = u_{p'} u_{q'}$. When $\mathcal{P}$ is input by the user the p and q fields of the X_j are set. The p' and q' fields are initialized so that $X_j = (p, q, 2j-1, 2j)$ See below.

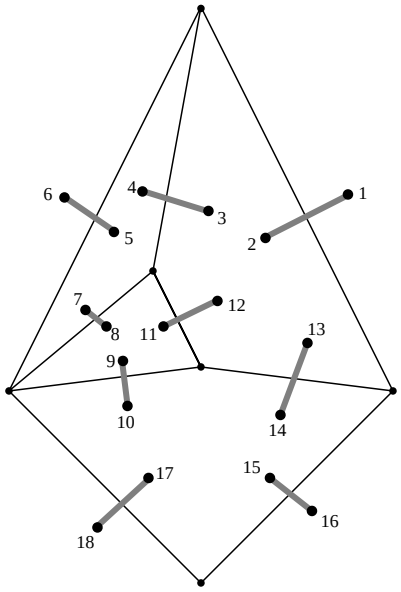

Fig. 2. The initial state of the example of Fig. 1.

Edges in $\mathcal{P}$ are isomorphic to edges in $\mathcal{P}'$ so that each edge has a dual edge. This is not the case for vertices, but I want to define the dual vertex of a vertex for the purposes of this algorithm. Given an edge $X_j = (p, q, p', q')$, the label of the dual vertex of a vertex in an edge is $\mathrm{dv}(v_p, e_j) = p'$ and $\mathrm{dv}(v_q, e_j) = q'$, and the label of the other dual vertex is by $\mathrm{odv}(v_p, e_j) = q'$ and $\mathrm{odv}(v_q, e) = p'$ (Fig. 3). In this way, each edge is arbitrarily oriented according to which vertices are considered the head and tail.

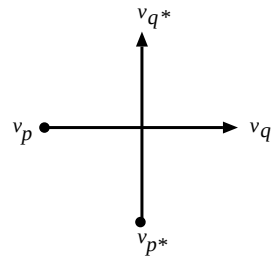

Fig. 3. Dual vertices.

To construct $\mathcal{P}'$, we first visit a vertex v_i in the interior of $\mathcal{P}$. Initially, v_i would look like the left side of Fig. 4. For every pair of edges e_j, e_k incident to v_i such that $e_k = \text{succ}(e_j)$, rename all occurrences of $\text{dv}(v_i, e_k)$ in the third or fourth fields of the the X_j to $\text{odv}(v_i, e_j)$. After the loop terminates, v_i would look like the right side of the figure. We continue this process for each interior vertex.

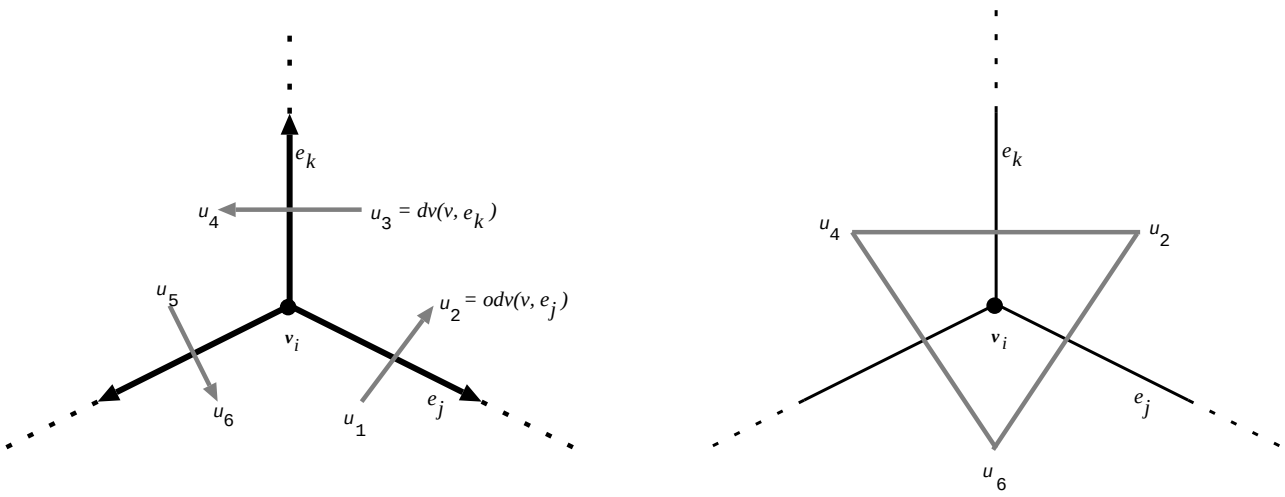

Fig. 4. Action of the algorithm at an interior vertex.

This algorithm works because even though $\text{dv}(v, e)$ arbitrarily assigns a dual vertex, it consistently considers v to be the tail of a directed edge and does so at each vertex.

To construct the poles of the dual we first connect the north and south poles of $\mathcal{P}$ with two edges. This new network is no longer a p–net but this procedure will result in the creation of the polar dual of the original p–net, $\mathcal{P}$. The edges will be denoted e_N (north edge) and e_S (south edge) and will be placed into the circular orderings C_N, C_S about the poles as on the right side of Fig. 5. The graphical input of $\mathcal{P}$ will be restricted at the north pole as in the left side of Fig. 5 (the north pole is at the origin). Similarly for the south pole. This will allow e_N and e_S to be inserted correctly into C_N by choosing the angles they make with v_N to be between $\frac{\pi-\delta}{2}$ and $\frac{\pi+\delta}{2}$. Similarly for C_S.

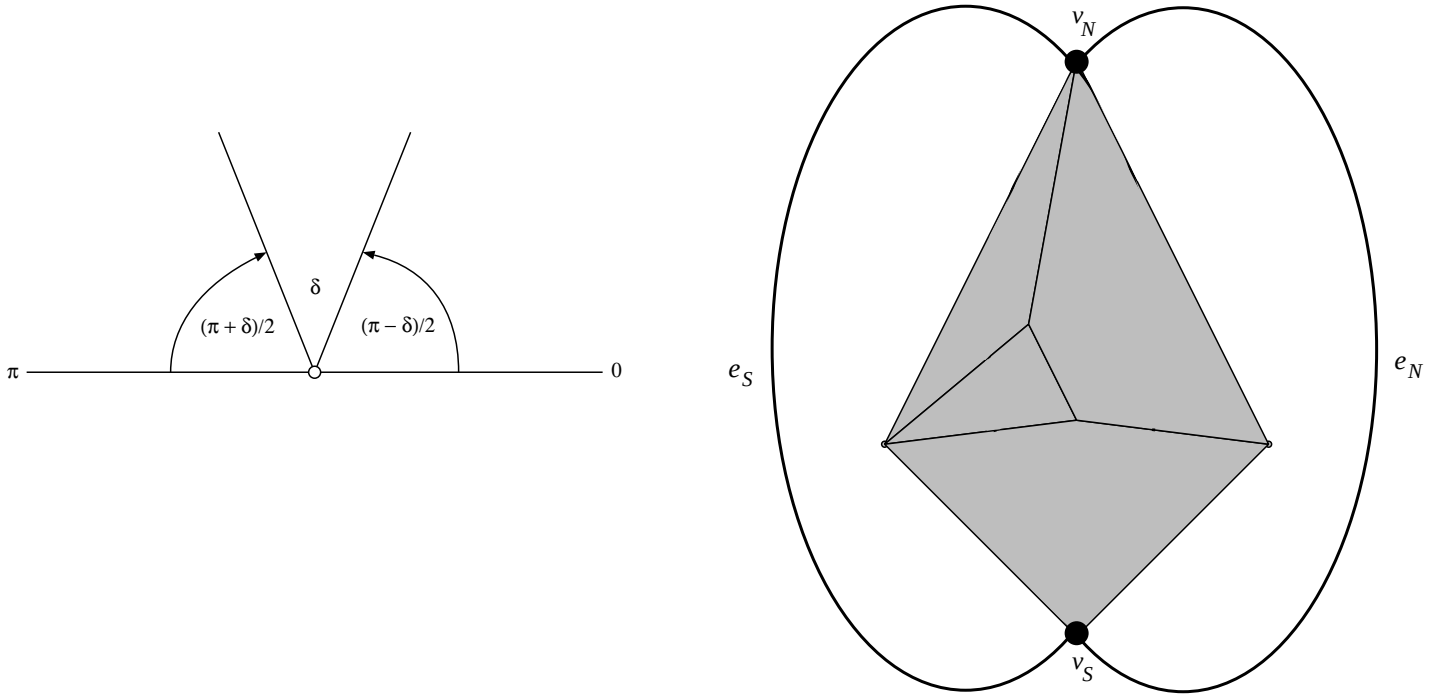

Fig. 5. Placement of e_N and e_S.

Now that e_N, e_S have been properly placed about the poles, we perform the same algorithm about v_N and v_S that was performed at interior nodes *except* when $e_S = \text{succ}(e_N)$ at v_N and $e_N = \text{succ}(e_S)$ at v_S. For our example, the result appears in Fig. 6 on the next page.

At this point $\mathcal{P}'$ has been constructed and all that remains is to identify $u_{N'}$ and $u_{S'}$. Initially, $e_N = (N, S, \alpha, \beta)$ and $e_S = (N, S, \omega, \varphi)$, where the Greek letters denote arbitrary vertex labels. After the algorithm has terminated, $u_{N'}$ will be u_α (recalling that $\{u_k\}$ are the vertices of $\mathcal{P}'$) because $\text{dv}(v_N, e_N) = \text{odv}(v_S, e_N) = \alpha$. Similarly, $u_{S'}$ will be u_φ.

Initially, $|\{u_k\}| = 2|\{v_i\}|$. When the algorithm is finished, many of the u_k will have been deleted. Let $U = \{u_{k_1}, u_{k_2}, \ldots, u_{k_m}\}$ be the final set of vertices and $\{e'_j\}$ be the final set of edges of $\mathcal{P}'$. The Kirchoff matrix of the dual is then

$$(K')_{ij} = \begin{cases} \deg u_{k_i}, & \text{if } i = j \\ -1, & \text{if } u_{k_i}u_{k_j} \in \{e'_j\} \\ 0, & \text{otherwise} \end{cases} .$$

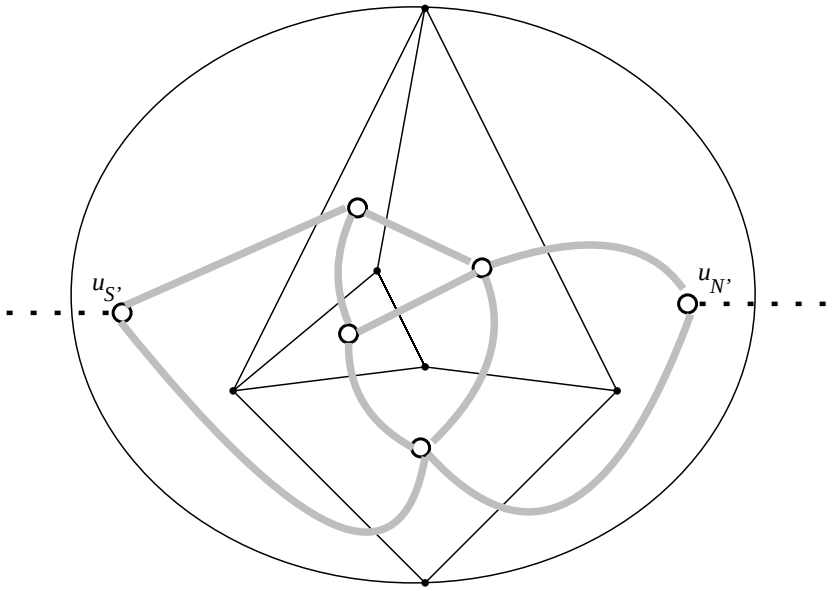

Fig. 6. Completely constructed dual.

5 Source Code

5.1 Dual.java

```
// Unfinished...

import java.*;
import Network;
import Edge;
import Matrix;

public class Dual
{
  private static final int maxfaces = 100;
  private static final int Graph = 0;

  public Network D; // Dual
  public int numEdges; // number of edges in N and NStar, number of squares
  pubic int northnode, southnode, northedge, southedge;
  public Edge[][] circOrder;
  public int[] incCount; // these two go together
```

```
  public int[][] adj;
  public int[] indices;
  public int[] imap;

  public Edge[] edges;
  public int numNodes, // number of vertices in N
    numFaces; // number of faces in N, #V* = numFaces+2
  public int numDualNodes;

  // will be a numEdges x 3 array such that coords[i][1] will be the x
  // coordinate of the upper left corner of the ith square,
  // coords[i][2] will be the y coordinate, and coords[i][3] will be
  // the length of the side of the square.  this will be accessed by
  // DrawNetwork when the user chooses to display the tiling
  public int[][] coords;

  public Dual(Network N, Point[] P)
  {
    // the N passed in already has the Dirichlet problem solved in it
    numNodes = N.numNodes();
    numEdges = edgecount(N);
    findNSnodes(); // initializes northnode and southnode
    initEdges(N); // prepare edges array
    getOrdering(P); // initializes and puts ordering in circOrdering[][]
    makeDual(); // stores results in edges[]
    solveDualDirichlet();
    makeTilingCoords(N);
  }

  public findNSnodes(Network N)
  {
    int node1, node2;

    for (int i = 0; i < numNodes; i++)
      if (N.onBoundary(i)) {
node1 = i;
break;
      }

    for (int j = i+1; j < numNodes; j++)
      if (N.onBoundary(j)) {
```

```
node2 = j;
break;
      }

    if (N.volts(node1) > N.volts(node2)) {
      northnode = node1;
      southnode = node2;
    } else {
      northnode = node2;
      southnode = node1;
    }
  }


  public edgecount(Network N)
  {
    int num = 0;

    // are arrays indexed starting at zero or one???
    // visit every entry in the upper triangular part of the adjacency matrix
    // and see if it is nonzero.
    for (int i=0; i < numNodes; i++)
      for (int j=i+1; j < numNodes; j++)
if (N.connected(i,j))
  num++;

    return num;
  }

  public initEdges(Network N)
  {
    edges = new Edge[numEdges+2];
    adj = new int[numNodes][numNodes];
    northedge = numEdges;
    southedge = numEdges+1;

    for (int i = 0; i < numNodes; i++)
      for (int k = 0, int j = i+1; j < numNodes; j++)
if (N.connected(i,j)) {
  edges[k].v1 = i;
  edges[k].v2 = j;
  edges[k].label = k; // looks superfluous, but isn't
```

```
  edges[k].connection = true;
  adj[i][j] = adj[j][i] = k;
  k++;
} else adj[i][j] = 0;

    edges[northedge].v1 = edges[northedge].dv1 = northnode;
    edges[northedge].v2 = edges[northedge].dv2 = southnode; // this will be N'
    edges[northedge].connection = true;
    edges[southedge].v1 = edges[southedge].dv1 = southnode;
    edges[southedge].v2 = edges[southedge].dv2 = northnode; // this will be S'
    edges[southedge].connection = true;
  }

  public getOrdering(Network N, Point[] P)
  {
    int i,j,k;
    double theta;
    circOrder = new Edge[numNodes][maxInc];
    incCount = new int[numNodes];

    for (i = 0; i < numNodes; i++)
      {
k = 0;
for (j = 0; j < numNodes; j++)
  if (N.connected(i,j) &&
      (!(i == northnode && j == southnode) &&
       !(i == southnode && j == northnode)))
    {
      theta = atan2(P[j].x-P[i].x, P[j].y-P[i].y)+PI;
      circOrder[i][k].theta = theta;
      circOrder[i][k].label = adj[i][j];
      circOrder[i][k].v1 = i;
      circOrder[i][k].v2 = j;
      k++;
    }
incCount[i] = k;
SortOnTheta(circOrder[i], incCount[i]);
      }

    // new Edge(edges[northedge]) ???
    circOrder[northnode][incCount[northnode]+1] = edges[northedge];
    circOrder[northnode][incCount[northnode]+2] = edges[southedge];
```

```
    circOrder[southnode][incCount[southnode]+1] = edges[southedge];
    circOrder[southnode][incCount[southnode]+2] = edges[northedge];
    incCount[northnode] += 2;
    incCount[southnode] += 2;
  }

  public SortOnTheta(Edge[] e, int num) // insertion sort
  {
    int lo = 0, up = num;
    int i, j;

    Edge temp;
    for ( i=up-1; i>=lo; i-- ) {
      temp = e[i];
      for ( j=i+1; j<=up && (temp.theta > e[j].theta); j++ )
e[j-1] = e[j];
      e[j-1] = temp;
    }
  }

  public makeDual()
  {
    connectDualEdges();
    numDualNodes = getNumDualNodes();
    makeIMap();
    makeDualNetwork();
  }

  public connectDualEdges()
  {
    int i,j,p,q;

    for (i = 0; i < numNodes; i++)
      for (j = 0; j < incCounts[i]; j++)
{
  p = circOrder[i][j].label;
  q = circOrder[i][(j+1) % incCounts[i]];
  if (!(p == northedge && q == southedge) &&
      !(p == southedge && q == northedge))
    updateDual(dv(i,p), odv(i,q));
}
  }
```

```
  public updateDual(int dv, int odv)
  {
    for (int i = 0; i < numEdges+2; i++) // +2 for boundary edges
      if (edges[i].dv1 == dv)
edges[i].dv1 = odv;
      else if (edges[i].dv2 == dv)
edges[i].dv2 = odv;
  }

  public int dv(int v, int j) { return edges[j].v1 == v ? dv1 : dv2; }
  public int odv(int v, int i) { return edges[i].v1 == v ? dv2 : dv1; }

  // N' := S, S' := N

  public makeIMap()
  {
    indices = new int[MAXSIZE];

    for (int i = 0; i < 2*numEdges; i+=2) {
      indices[i] = edges[i].dv1;
      indices[i+1] = edges[i].dv2;
    }

    sortIndices(indices, 2*numEdges);
    fillMapArray();
  }

  public sortIndices(int[] array, int num) // insertion sort
  {
    int lo = 0, up = num;
    int i, j;

    int temp;
    for ( i = up-1; i >= lo; i-- ) {
      temp = array[i];
      for ( j=i+1; j<=up && (temp > array[j]); j++ )
array[j-1] = array[j];
      array[j-1] = temp;
    }
  }
}
```

```
  public fillMapArray()
  {
    int imap = new int[numDualNodes];
    int j = 1;

    imap[0] = indices[0];
    for (int i = 1; i < 2*numEdges; i++)
      if (indices[i] != indices[i-1]) {
imap[j] = indices[i];
j++;
      }

    // must put make the boundary nodes the 0 and 1 entry of the array
    putBoundaryAtStart();
  }

  public int getNumDualNodes()
  {
    int x = new int[numNodes];
    int num = 0;

    for (int i = 0; i < numNodes; i++) x[i] = 0;

    for (i = 0; i < numEdges+2; i++) // +2 for boundary edges
      {
x[edges[i].dv1]++;
x[edges[i].dv2]++;
      }

    for (i = 0; i < numNodes; i++)
      if (x[i] != 0)
num++;

    return num;
  }

  // this takes the labels of the dual which are unordered to
  // {1,2,...,numDualNodes}
  public int getLabel(int n)
  {
    for (int i = 0; i < numDualNodes; i++)
```

```
      if (imap[i] == n)
return i;
  }

  public makeDualNetwork() {
    D = new Network;

    D.addNode(true); // N'
    D.addNode(true); // S'

    for (int i = 2; i < numDualNodes; i++)
      D.addNode(false);

    for (int j = 0; j < numEdges; j++)
      connect(getLabel(edges[i].dv1), getLabel(edges[i].dv2), 1.0);
  }

  public solveDualDirichlet() // Due to M. Hoefer
  {
    //this function determines interior voltages of network
    //put boundary voltages into network

    int n = D.numInterior();
    int m = D.numBoundary();
    //Create C
    Matrix C = new Matrix(n,n);
    for(int i=m;i<D.numNodes();i++)
      for(int j=m;j<D.numNodes();j++) {
if(i == j) {
  //check all nodes if they are neighbors
  for(int l=0;l<D.numNodes();l++)
    if(D.connected(i,l)) {
      C.set(i-m,j-m,C.get(i-m,j-m)+D.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(D.connected(i,j))
    C.set(i-m,j-m,-D.conductance(i,j));
}
```

```
      }
    //create b
    Matrix Bt = new Matrix(n,m);
    for(int i=m;i<D.numNodes();i++)
      for(int j=0;j<m;j++) {
if(i == j) {
  //check all nodes if they are neighbors
  for(int l=0;l<D.numNodes();l++)
    if(D.connected(i,l)) {
      Bt.set(i-m,j,Bt.get(i-m,j)+D.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(D.connected(i,j))
    Bt.set(i-m,j,-D.conductance(i,j));
}
      }
    Matrix phi = new Matrix(D.numBoundary(),1);
    for(int i=0;i<D.numBoundary();i++)
      phi.set(i,0,-D.volts(i));
    Matrix b = new Matrix(Bt.mult(phi));

    //since C is symmetric, positive definite, we can factorize 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=D.numBoundary();i<D.numNodes();i++)
      D.setVolts(i,x.get(i-D.numBoundary(),0));
    repaint();
  }


  public makeTilingCoords(Network N)
  {
    int C = Det(N);
    coords = new int[numEdges][3];

    // will be a numEdges x 3 array such that coords[i][0] will be the x
    // coordinate of the upper left corner of the ith square,
    // coords[i][1] will be the y coordinate, and coords[i][2] will be
    // the length of the side of the square.  this will be accessed by
    // DrawNetwork when the user chooses to display the tiling
    for (int i = 0; i < numEdges; i++)
      {
coords[i][0] = round(C *
     min(D.volts(getLabel(edges[i].dv1),
 D.volts(getLabel(edges[i].dv2)))));
coords[i][1] = round(C * max(N.volts(edges[i].v1),
     N.volts(edges[i].v2)));
coords[i][2] = round(C * N.current(edges[i].v1, edges[i].v2));
      }
  }
}
```

5.2 Edge.java

```
import java.*;

public class Edge
{
  public int v1, v2, label;
  public int dv1, dv2;
```

```
  public double theta;
  public boolean connection;

  public Edge()
  {
    theta = 0.0;
    label = noedge;
    v1 = v2 = dv1 = dv2 = 0;
    connection = false;
  }

  public boolean connected()
  {
    return (connection);
  }
}
```

References

[1] R. L. Brooks, C. A. B. Smith, A. H. Stone, W. T. Tutte, *The Dissection of Rectangles Into Squares*, Duke Math. J., 7, 312–340.

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

[3] M. Hoefer, *A Graphical User Interface for the Creation, Manipulation, and Calculation of Electrical Networks*, REU Report, 1996.