SURF Final Report
Hyperbolic versus Euclidean Embedding of Real-World Networks

Sadaf Amouzegar
Mentors: Elizabeth Bodine-Baron and Dr. Babak Hassibi
California Institute of Technology

October 31, 2011

Abstract

Recent research has suggested that social networks can be described in terms of a hidden metric space, which often can be used to improve applications requiring navigation, such as routing or search. Different algorithms exist to embed networks in either Euclidean or hyperbolic space. In this work we evaluate two representative algorithms to determine the optimal embedding for various types of networks. We use classical Multidimensional Scaling (MDS) as representative of Euclidean embedding. This method measures the length of the shortest path between any two nodes in the network and assigns a coordinate to every node in Euclidean space based on this distance. However, since Euclidean space has a geometric growth rate and most real-world networks exhibit low diameters, implying exponential growth, we also employ Crovella's "online greedy graph algorithm" as representative of hyperbolic embedding. This method uses a spanning tree algorithm to calculate the appropriate coordinates for each node in hyperbolic space. To determine which type of hidden space is optimal for different types of networks, we evaluate these algorithms on both real world and online social network data.

1 Introduction

Recent studies in network science have focused on the characteristics and development of social networks. Analysis of real-world data has revealed that these networks tend to share the same characteristics of high clustering, heavy-tailed degree distributions, small diameter, and searchability. In the 1960's, Stanley Milgram carried out several experiments in which subjects were asked to deliver letters to an unknown recipient simply by forwarding them to friends whom they thought would be acquainted with the target. Roughly 20% of the letters were delivered to the target recipient in an average of 6 steps. This experiment demonstrated the "small diameter" property of social networks, in which the diameter of (and paths between individuals in) the network scales logarithmically with the size of the network. More importantly, his experiments showed that in real-world networks, individuals could use a local greedy algorithm to discover these short paths through the network, a property called "searchability." Many individuals reported using geographical proximity as an important factor in their forwarding strategy, assuming that people living close to each other may have social contact. This assumption has led to several network models where the probability of links between nodes is directly influenced by their geographical proximity [5]. Further, recent research has suggested that social networks can be described in terms of a hidden metric space, which may capture the geographic proximity of nodes, or something more abstract, like social distance.

Hidden metric spaces have very important practical applications for search and routing in large complex networks. For example, classic geometric routing treats the geographic coordinates of the nodes as addresses in the process of routing. However, geometric routing is complicated due to two reasons: first, it is very costly to add GPS receivers to every node. Second, the precise coordinates of the nodes do not account for topological properties of networks (such as the "long-range" connections that are so common in social networks). To address these issues, current algorithms propose assigning virtual coordinates to nodes based

on their locations in a hidden space, which may or may not be their geographic location in the real world. Greedy routing algorithms can then use these virtual coordinates to send messages through the network.

This paper focuses on evaluating the difference between embedding networks in hyperbolic and Euclidean space. We implement two embedding algorithms, Multidimensional Scaling (MDS) [1] for embedding a network in Euclidean space, and an "online greedy graph algorithm" from [3] for embedding in hyperbolic space (specifically, the Poincaré disk). We then use these algorithms to embed several classic random networks, an Erdös-Rènyi random network, a small world network, and a preferential attachment network, in addition to several real-world social networks.

2 Methods

In this section, we describe in detail the algorithms we use to embed a given network in both hyperbolic space and Euclidean space.

2.1 Embedding in Euclidean Space

We use Multidimensional Scaling (MDS) to embed nodes in Euclidean space, since it is a relatively simple method for embedding a collection of nodes in $\mathbb{R}^d$. The input to MDS is an nxn distance matrix P and the output is an nxd matrix of coordinates such that the distance between any pair of nodes corresponds to the distances in P [1]. Formally, the MDS algorithm proceeds as follows [6].

Step 1 Create an "inner-product" matrix centered at the mean:

$$b_{ij} = -\frac{1}{2}\left(p_{ij}^2 - \frac{1}{n}\sum_{j=1}^{n} p_{ij}^2 - \frac{1}{n}\sum_{i=1}^{n} p_{ij}^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}^2\right)$$

Step 2 Calculate the eigenvalues and eigenvectors of B:

$$B = VAV^T$$

Step 3 Calculate the coordinates in Euclidean space:

$$X = V\sqrt{A}$$

2.2 Embedding in Hyperbolic Space

Multidimensional Scaling is an efficient way to embed any given real-world network into Euclidean space. However, because the Euclidean space of dimension k has a geometric growth rate, and some small-world graphs exhibit low diameters, implying an exponential growth rate, we also employ an algorithm to embed a network into hyperbolic space. We use Mark Crovella's "online greedy embedding graph algorithm" as representative of hyperbolic embedding [3]. This method uses a spanning tree algorithm to assign appropriate coordinates to every node in a Poincaré disk. The advantage of this algorithm is that it allows for the addition of new nodes in an online manner such that the coordinates of all other nodes do not change.

In order to represent the virtual coordinates of the embedded nodes in the hyperbolic plane, we assign complex numbers from the set $D = \{z \subset \mathbb{C}|\ |z| < 1\}$. The Euclidean circle $\partial D = \{z \subset \mathbb{C}|\ |z| = 1\}$ represents the boundary at infinity of the model. All points located on the circle are referred to as ideal points or points at infinity of D. Two points on the Euclidean circle determine a hyperbolic line in D. Given two ideal points $a = e^{i\alpha}$ and $b = e^{i\beta}$, the center of the Euclidean circle containing the hyperbolic line whose endpoints at infinity are a and b, and the corresponding radius are given by:

$$c = 1/m^* \tag{1}$$

$$R^2 = 1/|m|^2 - 1 \tag{2}$$

such that $m = \frac{(a+b)}{2}$ is the midpoint of the arch connecting a and b, and m^* is the complex conjugate of m.

The algorithm takes a graph G described by a set of nodes and the connections between them as its input. An embedding of a tree graph T which spans the given graph G is also an embedding of the graph G [5]. While any type of tree is suitable, the one suggested by Crovella is the "minimal-depth tree" [3]. In order to form the tree, we must first choose a root node from the network nodes. Each of the other nodes n then selects as parent node p_n the neighbor closest to the root note. Formally, the algorithm proceeds as follows [3].

Step 1 Assign a root node r of the tree:

1. A virtual coordinate $C(r)$ in the hyperbolic plane
2. The angles $\alpha_r = \pi$ and $\beta_r = 2\pi$ corresponding to the points at infinity on the Euclidean circle $a_r = e^{i\alpha_r}$ and $b_r = e^{i\beta_r}$

Step 2 For each node $n \subset G$:

1. Its parent p_n
   (i) sends $C(p_n)$, $\alpha_n = \alpha_p$ and $\beta_n = (\alpha_p + \beta_p)/2$ to n and
   (ii) updates $\alpha_p = \beta_n$
2. Node n
   (i) calculates c and R according to (1) and (2) given previously with $a_n = e^{i\alpha_n}$ and $b_n = e^{i\beta_n}$ as well as its own coordinate:
   $$C(n) = \frac{R^2}{[(C(p_n))^* - c^*]} + c \tag{3}$$
   (ii) updates $\alpha_n = \frac{(\alpha_n+\beta_n)}{2}$

This online embedding algorithm produces node coordinates without use of information about the physical locations of the nodes within the graph; it only uses the hop distances between nodes on the network.

3 Embedding complex networks

In this section we show the embeddings of the networks we consider, for both algorithms. We tested both algorithms on a network generated by preferential attachment, a small-world network, an Erdös-Rènyi random network, and most importantly real-world networks.

Many real-world networks have certain nodes that possess large number of connections and thus form the "hub" of the network while the other nodes have only a few connections. In our experiment, we generated such networks by a process of preferential attachment [2]. This model begins with two nodes that are connected by an edge. Every time a new node is added, it randomly chooses an existing node to connect to. The probability of such selection is proportional to the number of connections the node already has or in other words, its degree.

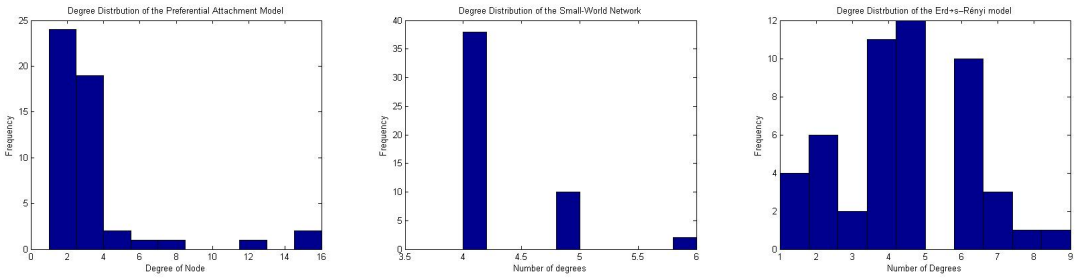
We also perform the embedding algorithms on a small-world network [7]. This is a type of graph in which the majority of nodes are not direct neighbors of one another but most nodes can be reached from any other node by a small number of steps. Mathematically, in a small-world network the distance L between any two nodes increases proportionally to the logarithm of the number of nodes N in the network, while maintaining high clustering. Three parameters must be considered in order to generate this model: the total number of nodes N, the degree of each node K, and the probability p of randomly replacing the edges. Initially, the model starts with N nodes, where every node is connected to K neighbors. The network is then randomized by rewiring each edge with probability p.

The Erdös-Rènyi model is used to generate random graphs such that an edge between any pair of nodes exists with equal probability [4]. For the purposes of our simulation, we generated graphs of 50 nodes, with each edge generated with probability p = 0.0790, independently from other edges.

Table 1 displays the empirical data sets and summarizes properties such as the number of nodes (N), the number of edges(E), the average clustering coefficient(CC), and the diameter(D) of the network graph.

Table 1: Experimental Data Sets

Network Model	N	E	CC	D
Preferential Attachment	50	76	0.1514	5
Small-World	50	214	0.4687	8
Erdös-Rènyi random network	50	219	0.1085	6



(a) Preferential Attachment (b) Small-world (c) Erös-Rènyi

Figure 1: Degree distributions for various network types.

3.1 Embedding by MDS

Here we give the results of embedding our various models of networks into a 2-dimensional Euclidean space via Multidimensional Scaling(MDS). Figures 2, 3, and 4 display Euclidean embeddings of the preferential attachment model, the small-world network, and the Erdös-Rènyi model, respectively.

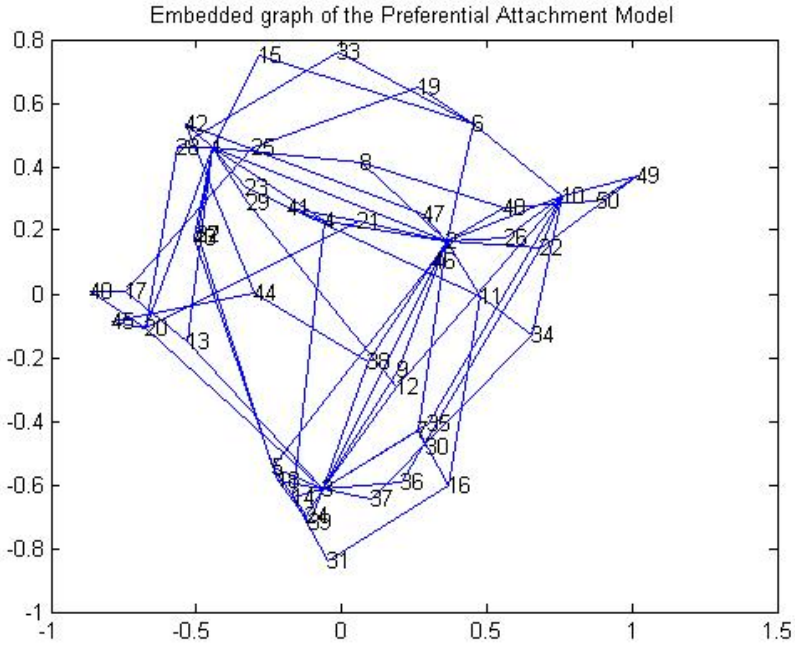


Figure 2: Euclidean embedding of preferential attachment model

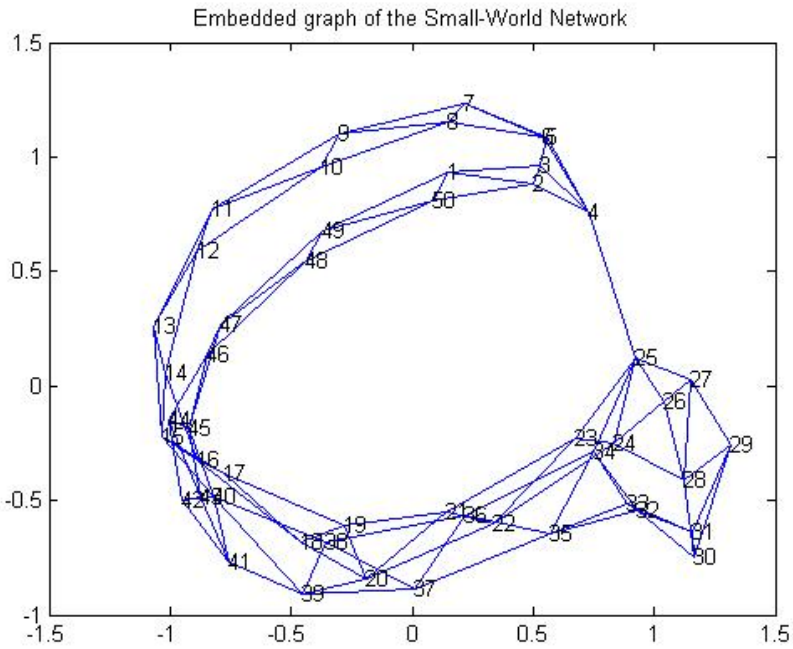


Figure 3: Euclidean embedding of small-world model

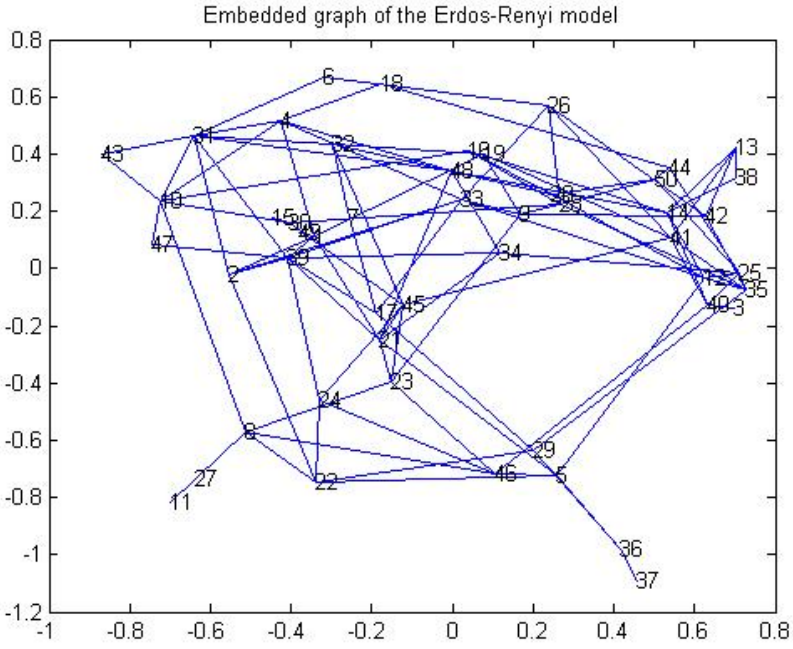
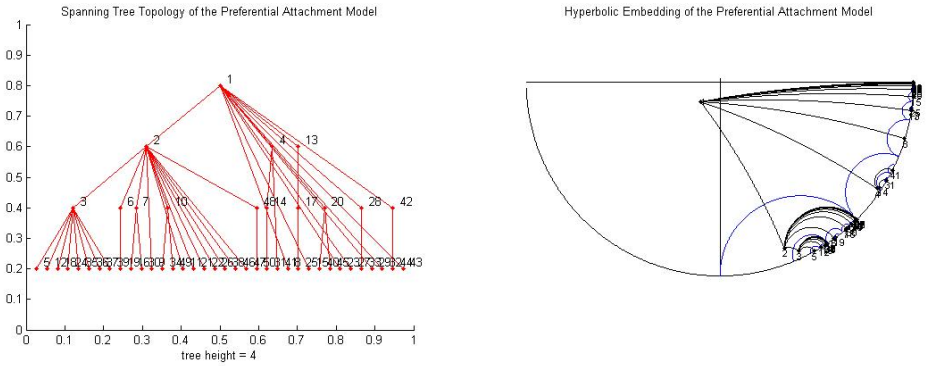


Figure 4: Euclidean embedding of ER model

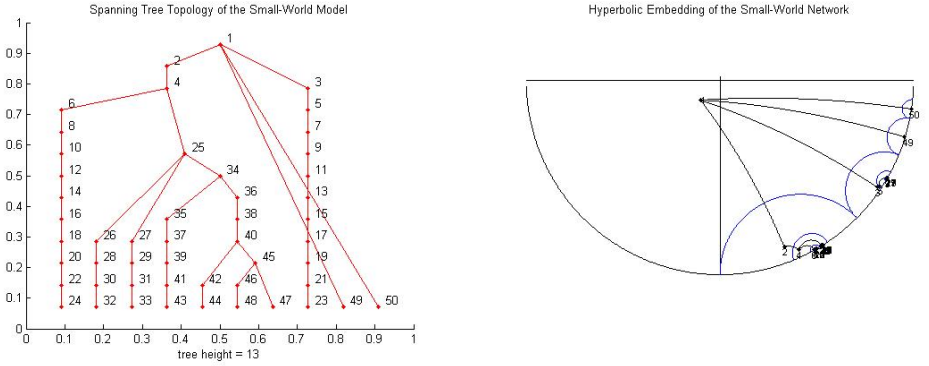
3.2 Embedding by the Online Greedy Algorithm

We also tested our algorithm for hyperbolic embedding on all three data sets. We first generated a spanning tree for each of the network graphs in order to compute a greedy hyperbolic embedding. Figures 5 - 7 show the graph of the spanning tree and the corresponding embedding into hyperbolic space for the preferential attachment model, the small-world network, and the Erdös-Rènyi model, respectively.



(a) Tree (b) Embedding

Figure 5: The spanning tree topology and the hyperbolic embedding of the preferential attachment model.

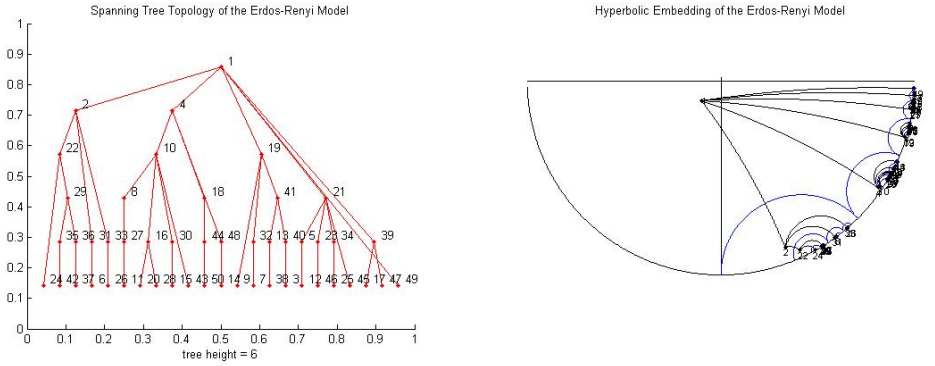


(a) Tree (b) Embedding

Figure 6: The spanning tree topology and the hyperbolic embedding of the small-world model.

4 Discussion and Future Work

In this project, we explored and tested two methods for graph embedding on three random networks generated from the preferential attachment, the small-world, and the Erdös-Rènyi models. The details of these data sets are shown in Table 1 and their degree distributions are displayed in Figure 1. Subjectively, the Euclidean embeddings of these data sets reflect their properties and structure. For instance, in Figure 2, the Euclidean embedding of the preferential attachment model clearly displays the various hubs and their many connections to neighboring nodes. The Euclidean embedding of the small-world network in Figure 3 reflects both the ring structure of the network and the property that most of the nodes can be reached within a small number of steps through the few random long-distance connections.



(a) Tree (b) Embedding

Figure 7: The spanning tree topology and the hyperbolic embedding of the ER model.

However, these properties are not as evident in the plots of their hyperbolic embeddings. This leads another important question - can we characterize the "hidden space" from the structure of the network? Is the topology of the hidden space even important? Perhaps, we may find that certain networks cannot be embedded in a particular hidden space, or than another type of space might be "better" for embedding. We can perform greedy routing using either Euclidean or hyperbolic space, but we might find that one space results in a higher success rate or a lower routing path length for a particular network.

Ultimately, we want to discover the optimal hidden space for any given real-world network. Is the more optimal hidden space the one with the more navigable topology? Or is it one that reproduces the common properties of complex networks: high clustering, small-diameter, and scale-free degree distributions? It remains as future work not only to define "optimal" but also to test our algorithms on real-world networks, such as social, transportation, neural, power-grid, citation, and other networks. Given any real-world data set, we can use the results of this paper to determine which type of hidden space is optimal for each type of network.

5 Acknowledgement

I would like to acknowledge and extend my heartfelt gratitude to my mentors Dr. Babak Hassibi and Elizabeth Bodine-Baron, whose guidance and support from the initial to the final level enabled me to develop an understanding of the subject and carry out my research. I would also like to thank Anrej Cvetkovski for assisting in writing the MATLAB codes for the online greedy algorithm.

References

[1] Xiaomeng Ban, Jie Gao, and Arnout Rijt. Navigation in real-world complex networks through embedding in latent spaces.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[3] Andrej Cvetkovski and Mark Crovella. Hyperbolic embedding and routing for dynamic graphs. In *Proc. of Infocom*, 2009.

[4] P. Erdös and A. Rènyi. On the evolution of random graphs. pages 17–61, 1960.

[5] R. Kleinberg. Geographic routing using hyperbolic space. *IEEE INFOCOM 2007 26th IEEE International Conference on Computer Communications*, pages 1902–1909, 2007.

[6] V. De Silva and J.B. Tenenbaum. Sparse multidimensional scaling using landmark points, 2004.

[7] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.

A Matlab Codes

A.1 Online Greedy Algorithm

```
%The algorithm presented here embeds any given connected graph into hyperbolic space
clear all;
close all;
clear classes;
format short;
format compact;

load network.mat %loading whatever random adjacency graph was generated via the artificial model
n =length(adj_mat); %total number of nodes in the graph

% Finding a minimal spanning tree of the adjacency matrix
adj_mat=sparse(adj_mat);
root=1; %Assigning a value of 1 to the root node
[distances_from_root discover_times tree]=f_mindepth_tree(adj_mat,root);

% Plotting the spanning tree topology
   figure;
   hold on;
   title('Spanning Tree Topology');
   xoffset=0.02; yoffset=0.02; fontsize=10;
   labels=1:n;
   f_treeplot_topology(tree,xoffset,yoffset,fontsize,labels);

% Rearranging labels
[sorted indeks]=sort(discover_times);

% Defining variables and formulas
    alpha(root)=pi;
    beta(root)=2*pi;
    coord(root)=-0.1-i*0.1; % we initialize the embedding process by assigning to the root
    % node a virtual coordinate in the hyperbolic plane along with two angles corresponding
    % to the ideal points on the Euclidean circle
    alpha_init(root)=alpha(root);% The variabes alpha_init is introduced because the algorithm
    % updates both angles of every node
    %The ideal points of the Euclidean circle with respect to the root node
    a(root)=exp(i*alpha(root));
    b(root)=exp(i*beta(root));
    %The center of the Euclidean circle and the corresponding radius
    geo_ctr(root)=2/( conj( a(root) + b(root) ) );
    geo_rad(root)=abs(geo_ctr(root) - a(root));

% Online Embedding Procedure
for ii=2:n,
   curnode=indeks(ii);
   parent=tree(curnode);

%  1)
   if alpha(parent) ~= alpha_init(parent),
     alpha(curnode) = alpha(parent);
     beta(curnode) = ( alpha(parent)+beta(parent) ) / 2;
   else %This step is executed if the parent node is the root node itself
     alpha(curnode) = ( alpha(parent)+beta(parent) ) / 2;
     beta(curnode) = ( alpha(curnode)+beta(parent) ) / 2;
   end %if
   alpha_init(curnode) = alpha(curnode);
   alpha(parent) = beta(curnode);

%  2) The geodesic center and radius with respect to the current node must be found
%  in order to calculate the coordinate of each node
```

```
    a(curnode) = exp(i*alpha(curnode));
    b(curnode) = exp(i*beta(curnode));
    geo_ctr(curnode) = 2 / ( conj( a(curnode) + b(curnode)) );
    geo_rad(curnode) = abs(geo_ctr(curnode)-a(curnode));

%   3) Calculating the coordinate of each node n
    coord(curnode) = geo_rad(curnode)^2/(conj(coord(parent)-geo_ctr(curnode)))+geo_ctr(curnode);
    alpha(curnode) = (alpha(curnode)+ beta(curnode)) / 2;

end

    % FIGURE FOR POINCARE DISK
       edgewidth=0.5;
       figure;
       hold on;
       figsize=1;
       figborder=0.02;
       title('Embedding');
       axis([-figsize figsize -figsize figborder]); axis equal; axis off;
       theta = (512:1024)/1024*2*pi;
       x = cos(theta);
       y = sin(theta);
       plot(x,y,'-k','linewidth',edgewidth);
       plot([-figsize,figsize],[0,0],'-k','linewidth',edgewidth);
       plot([0,0],[-figsize,figborder],'-k','linewidth',edgewidth);
       dotsize=7;  fontsize=7;

    % Plotting the root node
    curnode=root;
    plot(real(coord(curnode)),imag(coord(curnode)),'.k','markersize',dotsize);
    set(text(real(coord(curnode))+0.005,imag(coord(curnode))-0.005,num2str(curnode)),'FontSize',fontsize);

    % Plotting all other nodes along with their geodesics and edges to their parents
    for ii=2:n,

       % Plotting only the nodes
       curnode=indeks(ii);
       parent=tree(curnode);
       plot(real(coord(curnode)),imag(coord(curnode)),'.k','markersize',dotsize);
       set(text(real(coord(curnode))-0.007,imag(coord(curnode))-0.03,num2str(curnode)),'FontSize',fontsize);

       % Plotting the geodesics
       a1=angle(a(curnode)-geo_ctr(curnode));
       a2=angle(b(curnode)-geo_ctr(curnode));
       if real(geo_ctr(curnode))>0,
         if (a1>0) & (a2<0),a2=a2+2*pi;
         elseif (a1<0) & (a2>0), a1=a1+2*pi;
         end
         step=abs(a1-a2)/500;
         theta = min(a1,a2):step:max(a1,a2);
       else
         step=-abs(a1-a2)/500;
         theta = max(a1,a2):step:min(a1,a2);
       end
       zgeo = geo_ctr(curnode)+geo_rad(curnode)*exp(i*theta);
       plot(zgeo,'-b','linewidth',edgewidth);

       % Plotting the edges
       xa=real(coord(parent))+1.1e-8; ya=imag(coord(parent))+1e-8;
       xb=real(coord(curnode)); yb=imag(coord(curnode));
       aa=-(xb-xa)/(yb-ya); bb=(xa^2-xb^2+(ya-yb)*(ya+yb))/(2*(ya-yb));
         xc=(xa^2*yb-xb^2*ya+(ya-yb)*(ya*yb-1))/(2*(xa*yb-xb*ya));  yc=aa*xc+bb;  R=sqrt((yc-ya)^2+(xc-xa)^2);
       a1=angle(coord(parent)-xc-i*yc);  a2=angle(coord(curnode)-xc-i*yc);
       if xc>0,
         if     (a1>0) & (a2<0), a2=a2+2*pi;
```

```
        elseif (a1<0) & (a2>0), a1=a1+2*pi; end
        step=abs(a1-a2)/500;    theta = min(a1,a2):step:max(a1,a2);
      else
        step=-abs(a1-a2)/500;   theta = max(a1,a2):step:min(a1,a2);
      end
      xuc = R*cos(theta)+xc; yuc = R*sin(theta)+yc;
      plot(xuc,yuc,'-k','linewidth',edgewidth);
   end
```

A.2 Multidimensional Scaling

```
% Sadaf Amouzegar SURF Project 2011
% Code to embed network into Euclidean space using regular MDS
clear all;
close all;
% Load Erdos-Renyi network
load ERgraph.mat

k = 2; % the desired number of output dimensions
P = zeros(n);
B = zeros(n);

% Calculate shortest paths distances
D = all_shortest_paths(sparse(adj_mat));

% Classical MDS
for i = 1:n
   avg_dist(i) = mean(D(i,:));
end
avg = mean(avg_dist);

for i = 1:n
    for j = 1:n
        B(i,j) = -1/2*( D(i,j) - avg_dist(i) - avg_dist(j) + avg);
    end
end

[V,A] = eig(B);
lambda = diag(A);  % Note: in ASCENDING order
num_pos = length(find(lambda>0));
kpos = min(k,num_pos);

% Rearrange eigenvalues and vectors so that they are in DESCENDING order
lambda = sort(lambda,'descend');
for i = 1:n
    reversed(:,i) = V(:,n - i +1);
end
V=reversed;

for i = 1:kpos
    for j = 1:n
        X(i,j) = V(j,i) * sqrt(lambda(i));
    end
end

gplot(adj_mat,X')
title('Embedded graph, full MDS');
for K = 1:n
  text(X(1,K),X(2,K),num2str(K));
end
```