Graphs and Networks Lecture 24

The Last Lecture

Daniel A. Spielman December 2, 2010

24.1 Overview

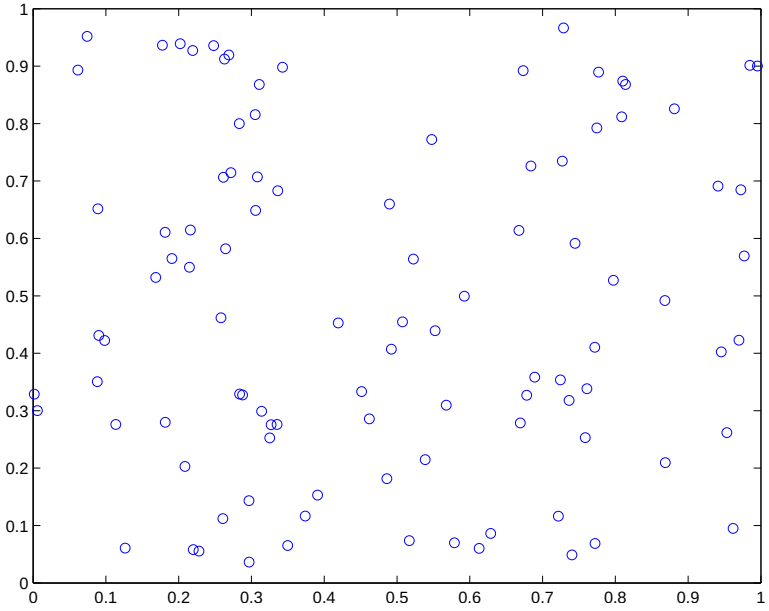
This lecture will address a few random topics that I'd like to mention before we end the semester.

24.2 Graph Drawing

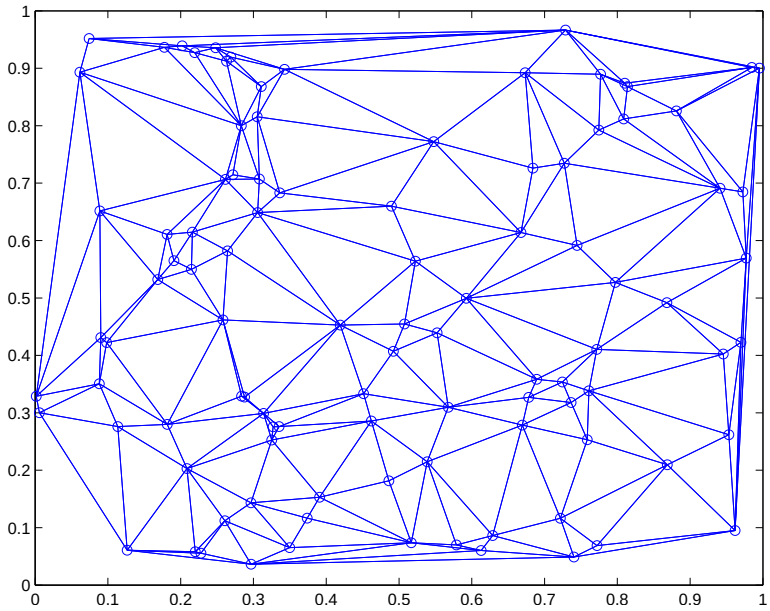
An amazing theorem of Tutte [Tut63] gives a method of drawing a 3-connected planar graph. Recall that a graph is 3-connected if there is no way of disconnecting it by removingly only 1 or 2 edges. Tutte showed how to construct a planar embedding of any planar graph by rubber bands. To begin, fix any face of the graph, and nail down the locations of the vertices on that face. Make sure to nail them down so that the edges form a convex polygon. Now, replace the edges by rubber bands, and let the vertices settle. Each non-nailed vertex will land at the center of gravity of its neighbors. Tutte proved that this is a planar embedding.

Let's do an example. First, I'll generate a random set of points in a box, and then create a planar graph on them by drawing a Delaunay triangulation.

```
>> [a,xy,tri] = delGraph(100);
>> clf
>> plot(xy(:,1),xy(:,2),'o');
```

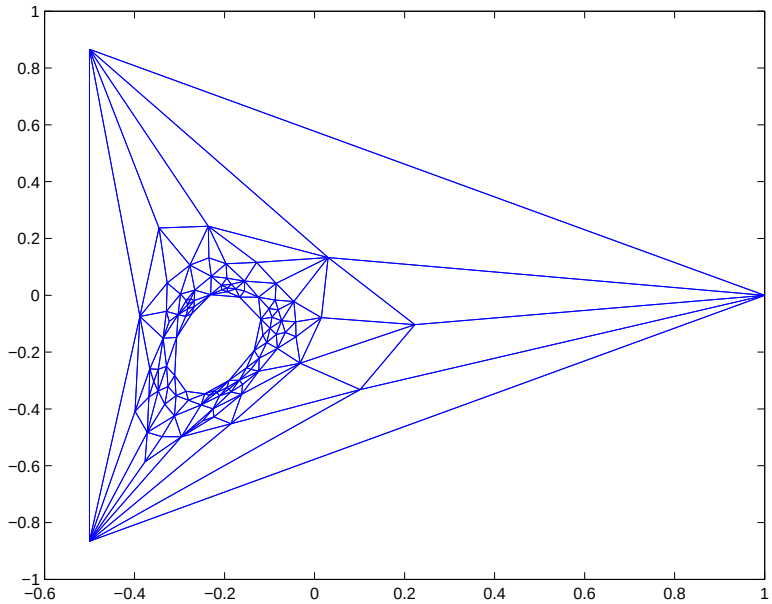


```
>> hold on
>> gplot(a,xy);
```



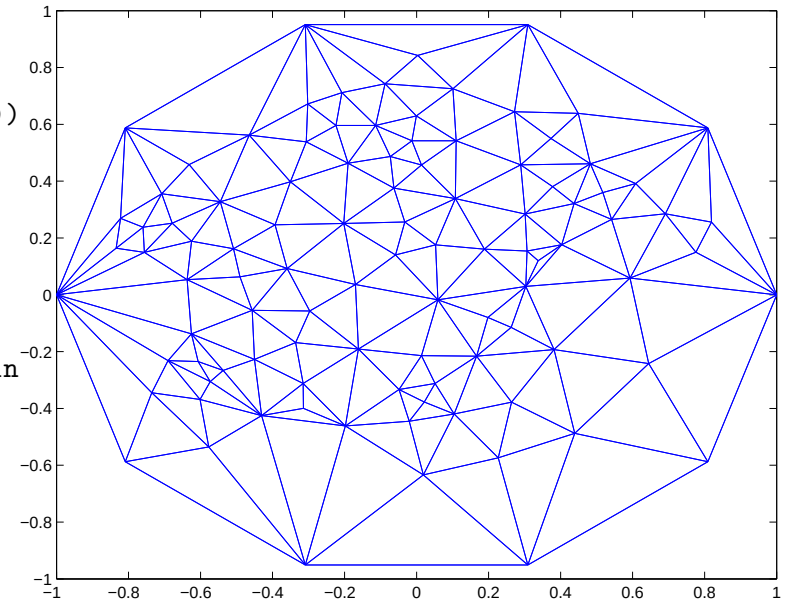
We will now pick an arbitrary triangle from this figure, nail down the locations of the vertices to be a regular triangle, and solve for the positions of the remaining vertices.

```
>> ind = tri(1,:)

ind =

    20    58    12

>> m = length(ind);
>> pos = [cos(2*pi*[1:m]/m);
sin(2*pi*[1:m]/m)]'

pos =

   -0.5000    0.8660
   -0.5000   -0.8660
    1.0000   -0.0000

>> la = diag(sum(a)) - a;
>> la(ind,:) = 0;
>> la(ind,ind) = eye(m);
>> b = zeros(length(a),2);
>> b(ind,:) = pos;
```



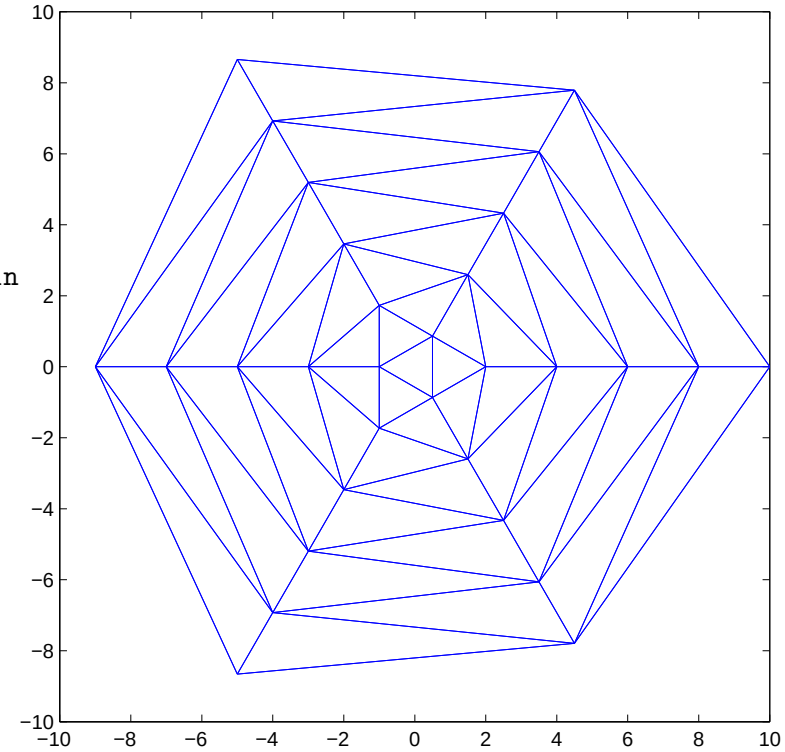
Of course, we would expect a better picture if we used an outside face instead of a triangle from the middle.

```
>> k = convhull(xy(:,1),xy(:,2))
>> ind = k(2:end);
>> m = length(ind);
>> la = diag(sum(a)) - a;
>> la(ind,:) = 0;
>> la(ind,ind) = eye(m);
>> b = zeros(length(a),2);
>> pos = [cos(2*pi*[1:m]/m); sin
>> b(ind,:) = pos;
>> xyt = la;
>> gplot(a,xyt)
```



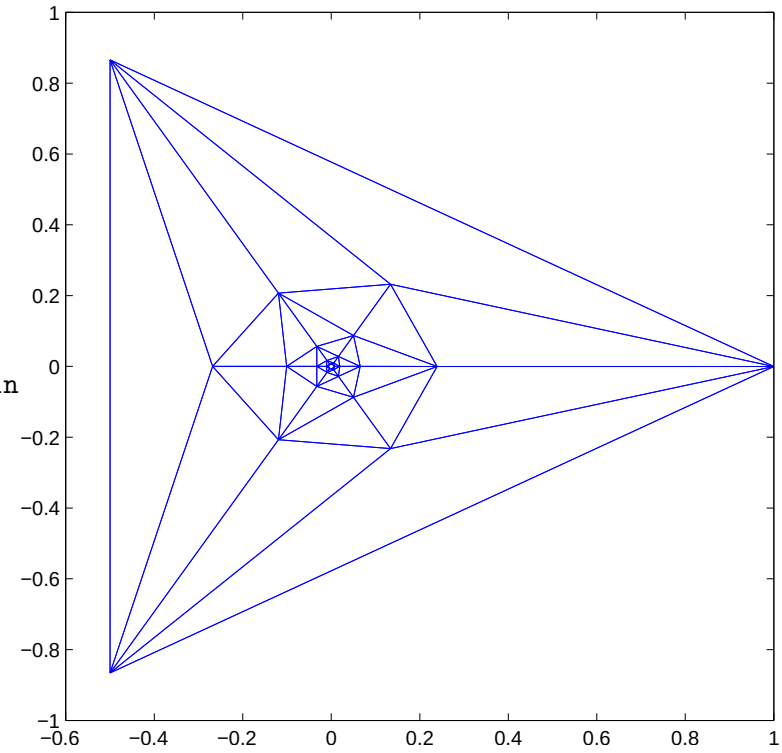
I don't want to give you the idea that these pictures are always good. For example, consider the Delaunay graph of triangles within triangles within triangles.

```
>> m = 3;
>> pos = [cos(2*pi*[1:m]/m); sin
>> for i = 1:10,
xy = [xy;pos * i * (-1)^i];
end
>> clf
>> plot(xy(:,1),xy(:,2),'o')
>> a = delGraph(xy);
```



When we draw it using Tutte's algorithm, we get triangles that become exponentially smaller.

```
>> ind = 1:3;
>> m = length(ind);
>> la = diag(sum(a)) - a;
>> la(ind,:) = 0;
>> la(ind,ind) = eye(m);
>> pos = [cos(2*pi*[1:m]/m); sin
>> b = zeros(length(a),2);
>> b(ind,:) = pos;
>> xyt = la;
>> gplot(a,xyt)
```



24.3 Graphs From Vectors

In the paradigmatic regressions and classification problems of Machine Learning, one is given a list of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ along with the values of some function on these vectors $y_1, \ldots, y_n$. One is then given a new vector $\boldsymbol{x}_0$ and asked to estimate the value of y_0. In the semi-supervised case, one is only given the values $y_1, \ldots, y_k$ for a k that is less than n. One of the major approaches to solving this problem is to associate a vertex with each vector, to connect them by a graph, and then to use learning algorithms on the graph.

There are many ways of choosing these graphs. Standard approaches either place edges between all pairs of nodes within some given distance, or construct k-nearest neighbor graphs. One can then weight the edges according to the distance between the vertices they connect. Standard choices of weighting functions are inverse in the distance, inverse in the square of the distance, exponential in the distance, and exponential in the square of the distance. For example,

$$e^{-d(\boldsymbol{x}_i, \boldsymbol{x}_j)^2/\sigma^2},$$

for some σ.

I believe that there should be a more systematic approach to doing this. I'll briefly tell you about an approach that Daitch and Kelner and I [DKS09] took to this problem. I don't think that it is the right approach. But, it might provoke the right approach.

We will build the graph just from the vectors $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n$, ignoring the labels. If we are going to have any chance of solving this problem, the function y will have to be related to the coordinates

of the vectors. So, we will try to build a graph that will help us solve for the coordinates of the vectors. To make this interesting, we will assume that we know the coordinates of every vector but one, and see how well we estimate that one. Our guess for the coordinates of a vertex i will be the weighted average of the coordinates of its neighbors:

$$\frac{1}{\sum_{(i,j)\in E} w_{i,j}} \sum_{(i,j)\in E} w_{i,j}\boldsymbol{x}_j.$$

We could then measure the error this provides:

$$\left\| \boldsymbol{x}_i - \frac{1}{\sum_{(i,j)\in E} w_{i,j}} \sum_{(i,j)\in E} w_{i,j}\boldsymbol{x}_j \right\|^2.$$

Our initial idea was to choose the graph that minimizes this over all choices for the vertex we leave out:

$$\sum_i \left\| \boldsymbol{x}_i - \frac{1}{\sum_{(i,j)\in E} w_{i,j}} \sum_{(i,j)\in E} w_{i,j}\boldsymbol{x}_j \right\|^2.$$

It turns out that this problem is degenerate in many ways. But, it becomes reasonable if we weight the contribution of each vertex by its weighted degree:

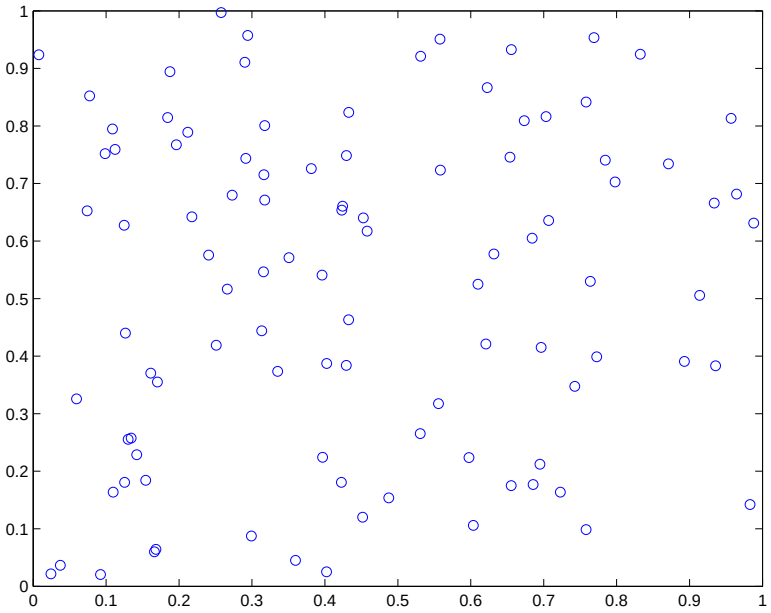
$$\sum_i \left\| \sum_{(i,j)\in E} w_{i,j}\boldsymbol{x}_j - \sum_{(i,j)\in E} w_{i,j}\boldsymbol{x}_i \right\|^2.$$

However, we now get zero if all edge weights are zero. So, we force a non-trivial solution by forcing every vertex to have weighted degree at least 1. The problem of computing the graph now becomes a semi-definite program, which we can solve.

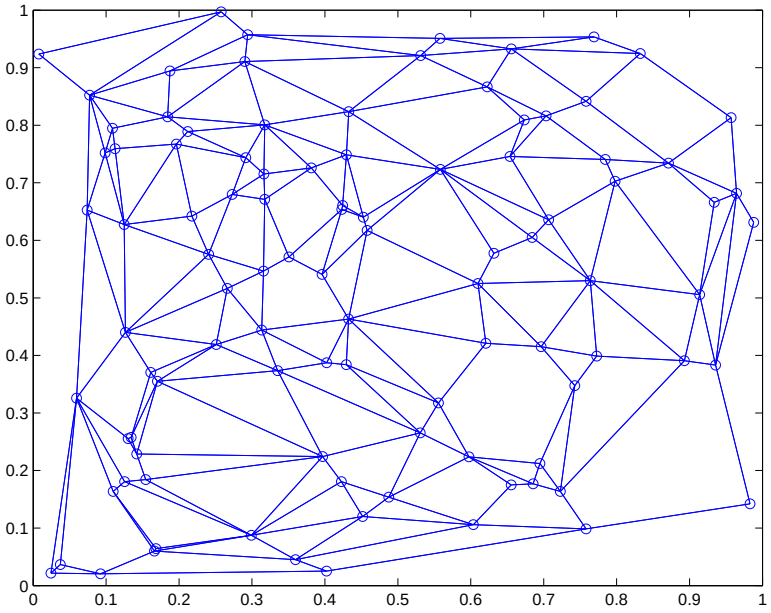
The resulting graph has some interesting properties. On many standard Machine Learning problems it gives better solutions than the other graphs. It also has interesting combinatorial properties. For example, for data lying in d dimensions it has average degree at most $d + 1$. For data lying in the plane, the graph is planar. That's pretty surprising.

Let's see an example. I'll choose 100 random points in the box, compute our graph, and then draw it.

```
>> x = rand(100,2);
>> plot(x(:,1),x(:,2),'o');
```




```
>> a = deg1graph(x');
>> gplot(a,x)
```



24.4 Respondent Driven Sampling

We will discuss the problem of Respondent Driven Sampling. It is an important problem to solve. But, the mathematical hurdles might be insurmountable. Either way, the claims of statistical validity reported in the literature are clearly unjustifiable.

24.5 Algebraically Defined Graphs

I will say a few words about the types of graphs we define mathematically.

References

[DKS09] Samuel I. Daitch, Jonathan A. Kelner, and Daniel A. Spielman. Fitting a graph to vector data. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 201–208, New York, NY, USA, 2009. ACM.

[Tut63] W. T. Tutte. How to draw a graph. *Proc. London Mathematical Society*, 13:743–768, 1963.