Social Resistance*

Michael P. Friedlander, Nathan Krislock, Ting Kei Pong

January 21, 2016

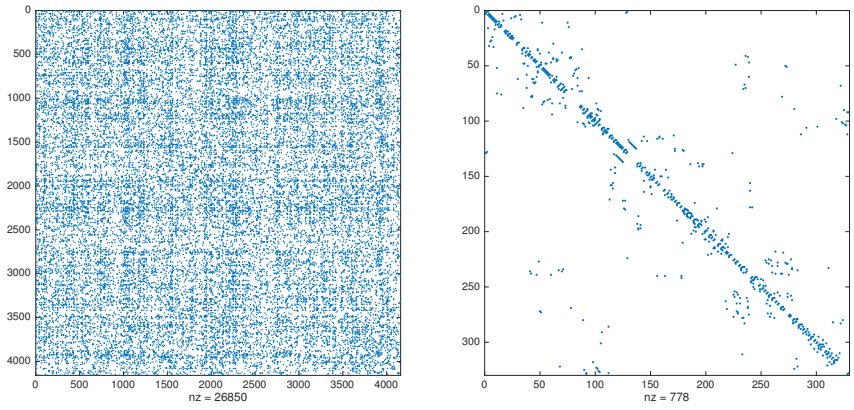

Figure 1: Collaboration (left) and traffic (right) networks.

Abstract.

Can we measure how close two people in a social network are just by knowing who is friends with whom? Can we infer the closeness of two people's research areas just by observing coauthor relations? Can we predict the importance of new roads just by looking at a current traffic network? In this case study, we investigate an approach to these questions and the algorithms behind the computations.

Keywords. Resistance distance, networks, rank-one Cholesky update, education

*This work was funded in part by a grant from the Carl Weiman Science Education Initiative at the University of British Columbia.

1 Introduction

This case study concerns an important question in social networks: how closely related two people are in the network. The notion of proximity that we use is the *resistance distance*. This notion of distance had its origin in the study of electrical networks [2]. In that context, it computes the resistance between the two points i and j in the network, assuming unit resistance on each edge. Its relationship with the information distance in social networks was recently identified [3, 1].

For large networks, this distance measure can be expensive to compute. As the network changes, it's useful to have an inexpensive way to make updates. We will study how various techniques for updating matrix factorizations can help to efficiently compute the resistance distances upon small changes in the network. We focus on Cholesky rank-one updates, and also briefly discuss the Sherman-Morrison-Woodbury formula (as one of the last exercises). We expect that you are familiar with the idea of the Cholesky factorization of positive definite matrices. See Trefethen and Bau [4, Lecture 23] for an accessible introduction to the Cholesky factorization. You will be guided through a derivation of the formula for the Cholesky rank-one update. There are two aspects of this exercise that may be new for you. First, you will have to figure out the vector used for the rank-one update when one additional connection in the network is identified. Second, when matrices are large, you will observe that there can be a big difference in the computational effort when the computation involves matrix-matrix multiplications rather than only matrix-vector multiplications.

As you read through this case study, you will be asked to work on a series of activities. These include short mathematical exercises, and short programming exercises for trying out these ideas on real data. You will need a working copy of MATLAB to complete the programming activities.

2 Background

In a social network, the question of how "close" are two people (or groups) is of great importance. The notion of closeness makes this question tricky. Take the network structure induced by Facebook friendship as an example. Such a structure can be conveniently modeled using an undirected graph. Mathematically, an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $\mathcal{V}$, and a set of edges $\mathcal{E}$ that describes connections between pairs of nodes. To model friendship on Facebook, one may represent an individual by a node and join two nodes with an edge if these two people are friends on Facebook. Figure 2 shows a small undirected graph representing the friendship of 6 individuals.

A natural measure of closeness in a graph is the length of the shortest path between two nodes. This is the least number of edges that need to be traversed in order to get from one node to another. Though well-studied mathematically, this notion is not an appropriate one when it comes to measuring how close two people are on a social network. For instance, according to this measure,

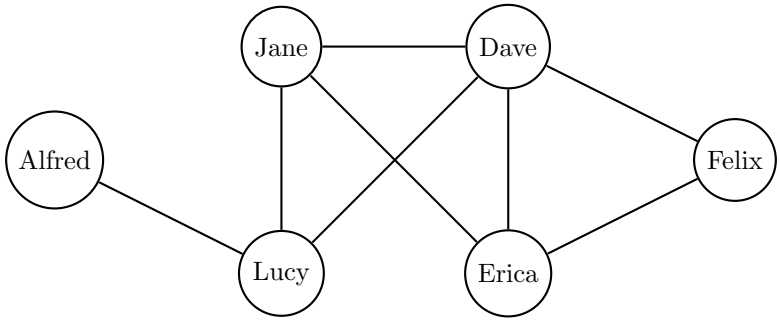

Figure 2: A small social network of 6 individuals.

Jane is equally close to Alfred and Felix in Figure 2. Intuitively, however, one would expect Jane to be closer to Felix because they have two common friends. Similarly, according to the shortest path measurement, as immediate friends, Dave is equally close to Jane and Felix. But again, one should expect Dave to be actually closer to Jane because they share two common friends. Intuition tells us that our closeness measurement should obey the following rule:

Two individuals are closer if they have more common friends.

The resistance distance conforms with this intuitive definition.

3 Resistance distance

To describe the resistance distance, we need more notions from graph theory.

We can conveniently represent an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes using an $n \times n$ symmetric matrix A which is called the *adjacency matrix* of the graph. The entries of this matrix are defined by

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{E}, \\ 0 & \text{otherwise,} \end{cases}$$

where we use (i, j) to denote an edge between nodes i and j, and the set $\mathcal{E}$ contains a list of all the edges in the graph. For example, the adjacency matrix of the graph in Figure 2 is given by

$$A = \begin{array}{c} \begin{array}{cccccc} \text{Alfred} & \text{Jane} & \text{Lucy} & \text{Dave} & \text{Erica} & \text{Felix} \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{array}{l} \text{Alfred} \\ \text{Jane} \\ \text{Lucy} \\ \text{Dave} \\ \text{Erica} \\ \text{Felix} \end{array},$$

and the sets of nodes and edges are given by

$$\begin{aligned}\mathcal{V} &= \{\text{Alfred, Jane, Lucy, Dave, Erica, Felix}\},\\ \mathcal{E} &= \{(\text{Alfred, Lucy}), (\text{Jane, Lucy}), (\text{Jane, Dave}), (\text{Jane, Erica}),\\ &\qquad (\text{Lucy, Dave}), (\text{Dave, Erica}), (\text{Dave, Felix}), (\text{Erica, Felix})\}.\end{aligned}$$

Note that this graph is *undirected*, i.e., it does not distinguish between an edge (i, j) and (j, i).

Let d be the vector whose ith entry is the degree of node i, i.e., the number of edges attached to node i. Then $d = Ae$, where e is the vector of all ones. In our example, $d = (1, 3, 3, 4, 3, 2)$. The Laplacian matrix L is defined as

$$L := \text{Diag}(d) - A,$$

where $\text{Diag}(d)$ is the diagonal matrix having the vector d along its diagonal. For the graph in Figure 2,

$$L = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 3 & -1 & -1 & -1 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}.$$

The Laplacian matrix a symmetric matrix and is always positive semidefinite; that is, $x^T Lx \geq 0$ for all vectors x.

A graph is called *connected* if any two nodes in the graph are connected by a set of edges. For example, the graph in Figure 2 is connected; however, if the edges (Jane, Lucy) and (Dave, Lucy) are removed, then the resulting graph is not connected since there would be no path from Alfred to Felix, for example. The nullspace for the Laplacian matrix of a connected graph is the set of vectors that are constant, i.e.,

$$Lv = 0 \quad \text{if and only if} \quad v = \lambda e$$

for some $\lambda \in \mathbb{R}$. You will explore these properties in Activity 1, below.

Activity 1

1. Prove that
$$L = \sum_{(i,j)\in E} \left(e^{(i)} - e^{(j)}\right)\left(e^{(i)} - e^{(j)}\right)^T, \tag{1}$$
where $e^{(i)}$ is the vector that is one in the ith entry, and is zero in every other entry.

2. Use the relationship (1) to show that the Laplacian matrix is positive semidefinite.

3. Show that $Le = 0$.

4. Let G be a connected graph and let L be its Laplacian matrix. Show that $L + ee^T$ is positive definite, i.e., that $x^T(L + ee^T)x > 0$ for all nonzero vectors x.

The resistance distance between nodes i and j on a connected graph is defined as

$$r_{ij} := \left\|e^{(i)} - e^{(j)}\right\|_M^2,$$

where $M = (L+ee^T)^{-1}$, and $\|x\|_M$ is the weighted 2-norm of a vector x, defined by

$$\|x\|_M = \sqrt{x^T M x}.$$

As required for $\|\cdot\|_M$ to be a valid vector norm, M is positive definite because $L+ee^T$ is positive definite. Note that the usual 2-norm of a vector corresponds to $M = I$, i.e., $\|x\| \equiv \|x\|_I = \sqrt{x^T x}$.

Activity 2

1. For the network in Figure 2, compute the resistance distances between Alfred and Jane, Alfred and Felix, Dave and Jane, as well as Dave and Felix. Does this notion of distance conform with our intuition?

Our next activity computes the resistance distances in a larger network: an author collaboration network on arXiv on the subject of "General Relativity". The data comes in the form of an adjacency matrix: the (i, j)th entry is 1 if person i and person j are coauthors, and is zero otherwise.

Activity 3

1. Download the matrix file `Graph.mat` (taken and modified from the UF Sparse Matrix Database). This matrix is an adjacency matrix of the collaboration network in Figure 1. Use the following MATLAB commands to load and view the matrix, as in Figure 1.

```
load Graph
spy(A);
```

2. Let $L + ee^T = U^T U$ be the Cholesky factorization of $L + ee^T$, where U is upper triangular. Show that the resistance distances can be computed as

$$r_{ij} = \left\|U^{-T}(e^{(i)} - e^{(j)})\right\|^2,$$

where $U^{-T} = (U^{-1})^T = (U^T)^{-1}$. Based on this, write a MATLAB function to compute r_{ij} for any given i and j. To compute $z = U^{-T}(e^{(i)} - e^{(j)})$ efficiently, you should solve the *lower-triangular* system $U^T z = b$, with $b = e^{(i)} - e^{(j)}$, using forward substitution; this is done in MATLAB using the command `z = U'\b`. Do not be tempted to use the MATLAB command `z = inv(U')*b`! This is less efficient and less accurate.

```
[rij,U] = Resistance(Lones,i,j);
```

The inputs are the positive definite matrix `Lones` (corresponding to $L+ee^T$ in the problem), i and j. The first output argument `rij` is the resistance distance r_{ij}, and the second output argument `U` is the Cholesky factor of `Lones`. You may use the MATLAB built-in function `chol` to compute U.

You may compare your output against `Resistance_soln.p`.

3. Download and complete `changeResistance.m`.

```
r = changeResistance(A,i,j,k);
```

This code takes in the adjacency matrix A, i and j, and also a positive integer k. It connects i to at most k random neighbors of j one by one, and recomputes the resulting resistance distance between i and j after each addition. The k computed resistances are recorded in the output vector r.

Experiment with the values $i = 500$, $j = 3000$ and $k = 10$. You may compare your output against `changeResistance_soln.p`. You can use `rng('default')` to reset the random seed for comparing the two codes.

Plot the decrease in the resistance distance as k increases. This indicates, intuitively, if researcher i works with coauthors of researcher j, then i and j get "closer" to each other on this collaboration network.

4 Rank-one updates

In the above activity, we performed a Cholesky factorization each time a collaboration is added. The overall work can be significantly reduced by using a Cholesky rank-one update, as we describe next.

Suppose that B is an $n \times n$ positive definite matrix and $B = U^T U$ is its Cholesky factorization. Given this upper triangular matrix U and a vector x, we would like to obtain a Cholesky factorization for $B + xx^T$. Note that

$$B + xx^T = U^T U + xx^T = [U^T \ x] \begin{bmatrix} U \\ x^T \end{bmatrix},$$

where the matrix

$$\begin{bmatrix} U \\ x^T \end{bmatrix} \tag{2}$$

is of size $(n+1) \times n$.

Activity 4

1. Let $(a, b) \in \mathbb{R}^2$ be given with $a > 0$. Find (c, s) with $c \geq 0$ and $c^2 + s^2 = 1$ so that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sqrt{a^2 + b^2} \\ 0 \end{pmatrix}. \tag{3}$$

The matrix (3) is an example of a Givens rotation matrix. One way to perform a Cholesky rank-one update is to apply successive Givens rotation matrices Q to "zero out" the last row of the matrix in (2) step by step, and obtain a new U in the process that is the Cholesky factor of $B + xx^T$.

Using the formulas you discover in Activity 4, you can find (c, s) with $c \geq 0$ and $c^2 + s^2 = 1$ such that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{pmatrix} U_{11} \\ x_1 \end{pmatrix} = \begin{pmatrix} \sqrt{U_{11}^2 + x_1^2} \\ 0 \end{pmatrix}.$$

Then let Q_1 be the Givens matrix

$$Q_1 = \begin{bmatrix} c & 0 & -s \\ 0 & I_{n-1} & 0 \\ s & 0 & c \end{bmatrix}$$

where I_{n-1} is the $(n-1) \times (n-1)$ identity matrix, and 0 is used to represent a vector or matrix of zeros with the appropriate dimension. Multiply the matrix in (2) by Q_1, and "zero out" the first entry of the last row:

$$Q_1 \begin{bmatrix} U \\ x^T \end{bmatrix} = Q_1 \begin{bmatrix} U_{11} & * \\ 0 & * \\ x_1 & * \end{bmatrix} = \begin{bmatrix} \sqrt{U_{11}^2 + x_1^2} & * \\ 0 & * \\ 0 & * \end{bmatrix} =: \begin{bmatrix} U^{(1)} \\ \left(x^{(1)}\right)^T \end{bmatrix}.$$

The symbol $*$ is used as a wild card to denote a possibly nonzero entry. Because of the position of I_{n-1} in Q_1, it only affects the first and last rows of the matrix that it multiplies.

Activity 5

1. Verify that Q_1 is orthogonal—that is, show that $Q_1^T Q_1$ is the identity matrix of size $(n + 1)$. Then show that

$$\begin{bmatrix} U^{(1)} \\ \left(x^{(1)}\right)^T \end{bmatrix}^T \begin{bmatrix} U^{(1)} \\ \left(x^{(1)}\right)^T \end{bmatrix} = B + xx^T.$$

A matrix Q_2 is then similarly chosen to operate on the second and last rows of

$$\begin{bmatrix} U^{(1)} \\ \left(x^{(1)}\right)^T \end{bmatrix}$$

so as to zero out the *second* entry of the last row, and so on. Thus, after k iterations we have

$$\begin{bmatrix} U^{(k)} \\ \left(x^{(k)}\right)^T \end{bmatrix} = Q_k \cdots Q_2 Q_1 \begin{bmatrix} U \\ x^T \end{bmatrix}.$$

Download and complete `mycholupdate.m`, which takes the above approach to compute Cholesky factor of $B + xx^T$, given U and x.

```
U = mycholupdate(U,x);
```

Remember, you do *not* need to form the whole Givens matrix; in each iteration you only need to find (c, s) with $c \geq 0$ and $c^2 + s^2 = 1$ such that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{pmatrix} U_{ii}^{(i-1)} \\ x_i^{(i-1)} \end{pmatrix} = \begin{pmatrix} \sqrt{\left(U_{ii}^{(i-1)}\right)^2 + \left(x_i^{(i-1)}\right)^2} \\ 0 \end{pmatrix}.$$

Test your code with random instances:

```
B = randn(3000); B = B'*B;
U = chol(B); x = randn(3000,1);
```

You may compare your code against `mycholupdate_soln.p`. After that, you may also take a look at the MATLAB built-in function `cholupdate.m`.

2. Download and complete `changeResistance_r1.m`:

```
r = changeResistance_r1(A,i,j,k);
```

It has the same inputs and outputs as `changeResistance.m`, and it uses the Cholesky rank-one update in the loop instead of computing Cholesky factors afresh. Guided by Activity 1.1, you will need to figure out what the x is when an edge is added to the graph.

Experiment with your code using $i = 500$, $j = 3000$ and $k = 10$. You may compare your output against `changeResistance_r1_soln.p`. How does it compare with the method in Activity 3.3 in terms of speed? You may also try to use the MATLAB built-in function `cholupdate.m` in place of `mycholupdate.m`.

5 The Sherman-Morrison-Woodbury Formula

Another widely used method for obtaining $(B + xx^T)^{-1}$ given B^{-1} is the Sherman-Morrison-Woodbury (SMW) formula. The SMW formula states that

$$(B + xx^T)^{-1} = B^{-1} - \frac{1}{1 + x^T B^{-1} x} B^{-1} x x^T B^{-1}.$$

The formula is particularly useful in our scenario because we only need to find the matrix-vector product $(B + xx^T)^{-1} v$ for a suitable B, x and v.

Activity 6

1. Argue that if B^{-1} is given, then computing $(B+xx^T)^{-1}v$ for given vectors x and v using the SMW formula only requires at most Cn^2 additions and multiplications, for some constant C independent of n.

2. Download and complete `chol_vs_smw.m`, which computes

$$y = (L + ee^T + xx^T)^{-1}(e^{(i)} - e^{(j)}),$$

where x corresponds to adding an edge between node i and a random neighbor of j:

```
[errchol,errsmw] = chol_vs_smw(A,i,j);
```

In this code, we assume knowledge of the Cholesky factor U of $L + ee^T$, and compare the additional cost needed to obtain y from performing the Cholesky update or using the SMW formula. The outputs `errchol` and `errsmw` are residuals

$$\|(L + ee^T + xx^T)y - (e^{(i)} - e^{(j)})\|,$$

with the y obtained from the corresponding approach.

You may compare your code against `chol_vs_smw_soln.p`. You may also try to use MATLAB's built-in function `cholupdate.m` in place of `mycholupdate.m`.

The SMW formula is convenient, but for some matrices, it can be *unstable*—that is, roundoff errors intrinsic to a computer's finite-precision arithmetic can overwhelm the final calculated answer [5]. It doesn't happen often, but it's something to bear in mind.

In our last activity, we are going to apply what we learned above to analyze the traffic network shown in the panel on the right-hand side of Figure 1.

Activity 7

1. Download the matrix file `Graph2.mat` (taken and modified from the UF Sparse Matrix Database). This is the adjacency matrix of the traffic network in Figure 1, and represents a subset of the road network of California. Load and view the matrix as in Activity 3.1.

2. Compute the resistance distances for all pairs of nodes. What is the average resistance distance across the whole network? Plot a histogram of the resistance distances.

3. As a greedy approach to decrease the average resistance distance, one can build a new road between the pair of nodes i and j that have the largest resistance distance.

   What is the average resistance distance across the whole network after such a new road is built? You can use either the Cholesky rank-one update or

the SMW formula to compute the new resistance distances. You may also want to test the effectiveness of the greedy approach by comparing the resulting average resistance distance with that obtained by adding a new road between two randomly chosen nodes.

6 Conclusion

We have only touched on a few of the basic methods in numerical linear algebra needed to analyze networks. With just a few refinements, it's possible to apply the techniques that you have learned to much larger networks. Can you spot opportunities to make your code more efficient? For example, in Activities 3 and 6 you computed the Cholesky factorization of $L + ee^T$ by first explicitly forming this matrix. But the very same techniques you studied for updating the Cholesky factorization could be used: first compute the Cholesky factorization $U^T U = L$, and then obtain the required factorization of $L + ee^T$ via `U = mycholupdate(U,e)`.

We also haven't touched on a crucial property of most networks that arise in practice: they have relatively few edges as compared to nodes, which means that the adjacency and Laplacian matrices that we studied are *sparse*—i.e., they have very few nonzero entries. The traffic network from which Figure 1 is extracted exemplifies this property. The original network, which has $19,171,281$ nodes, is too large to treat with the dense-matrix techniques used in this case study, but could be easily stored and manipulated by taking advantage of sparsity.

Numerical linear algebra is a cornerstone of countless practical problems, both old and new. We hope this case study has piqued your interest!

Author biographies

Michael P. Friedlander is Professor of Computer Science and Mathematics at the University of British Columbia, and Professor of Mathematics at the University of California, Davis. He received his PhD in Operations Research from Stanford University in 2002, and his BA in Physics from Cornell University in 1993. From 2002 to 2004 he was the Wilkinson Fellow in Scientific Computing at Argonne National Laboratory. He has held visiting positions at UCLA's Institute for Pure and Applied Mathematics (2010), and at Berkeley's Simons Institute for the Theory of Computing (2013). He serves on the editorial boards of SIAM J. on Optimization, SIAM J. on Matrix Analysis and Applications, SIAM J. on Scientific Computing, and Mathematical Programming. His research is primarily in developing and implementing numerical methods for large-scale optimization. Contact him `mpf@cs.ubc.ca`.

Nathan Krislock is an assistant professor at Northern Illinois University. His research interests include continuous and combinatorial optimization, with a focus on semidefinite programming, numerical computation, and applications. He received a PhD in the Department of Combinatorics & Optimization

from the University of Waterloo in 2010. He was a postdoctoral fellow at INRIA in Grenoble, France from 2010 to 2012, and then a PIMS postdoctoral fellow at the University of British Columbia from 2012 to 2013. Contact him at nkrislock@niu.edu.

Ting Kei Pong is an assistant professor at the Hong Kong Polytechnic University. He received his PhD in Mathematics from University of Washington in 2011. He was a postdoctoral fellow at University of Waterloo from 2011 to 2013, and was then a PIMS postdoctoral fellow at University of British Columbia from 2013 to 2014. His research is mainly on continuous optimization, with a focus on first-order methods for large-scale problems. Contact him at tk.pong@polyu.edu.hk.

Acknowledgments

The authors extend sincere thanks to Dianne O'Leary and Nargess Memarsadeghi for their steady encouragement and their many thoughtful suggestions for improving the presentation.

References

[1] Enrico Bozzo and Massimo Franceschet. Resistance distance, closeness, and betweenness. *Social Networks*, 35(3):460 – 469, 2013.

[2] D. Klein and M. Randic. Resistance distance. *Math. Chem.*, 12:81 – 95, 1993.

[3] Karen Stephenson and Marvin Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11(1):1 – 37, 1989.

[4] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. SIAM, 1997.

[5] EL Yip. A note on the stability of solving a rank-p modification of a linear system by the sherman-morrison-woodbury formula. *SIAM J. Sci. Stat. Comput.*, 7(2):507–513, 1986.