CS4220 Project 1

Due: February 28, 2015

All Roads Lead to Fresno

Task 1

MATLAB's $M\backslash e_1$ on my computer uses approximately 6.8 seconds to solve.

Task 2

MATLAB's LU decomposition of a sparse matrix M has P for row permuation and Q for column re-ordering such that

$$PMQ = LU.$$

But in order to maintain the *diagonal dominance* of sparse matrix M, the column re-ordering in this case always follows the row permutation. For instance, if row j of M is moved to row k, then the row-j vector of P is $P(j,:) = e_k^T$. In the column re-ordering matrix Q, in order to maintain the diagonal dominance of the matrix structure, column j is moved correspondingly to column k, which requires the column-j vector to be $Q(:,j) = e_k$. Consequently,

$$P(j,:) \cdot Q(:,j) = e_k^T e_k = 1.$$

For any rows which are not permuted, $P(s,:) = e_s^T$ and $Q(:,s) = e_s$. (In other words, P and Q start as identity matrices before permutations and column re-ordering). When putting together P and Q according to the scheme outlined above, we have found $PQ = I$, hence P and Q are orthogonal. (It has been verified in MATLAB.)

Task 3

Argue that N is symmetric and positive definite.

- Because A is symmetric, α is a constant, so $N = D - \alpha A$ is also symmetric.
- Since

  $$T = AD^{-1},$$
  $$N = D - \alpha A = (I - \alpha T)D = MD,$$

  where $D = \mathrm{diag}(d_1, d_2, \ldots, d_n)$, and d_j is the degree at node j, we can show the following by looking into how matrix N is constructed:

a. $N_{kk} = d_k > 0$: because the adjacency matrix A has all zero diagonal entries ($A_{kk} = 0, k = 1, \ldots n$), it means N has all positive diagonal entries.

b. For every row i (or column i, since N is symmetric), we have $N_{ii} = d_i$, and that the i-th row sum (except the diagonal term) $\sum_{j \neq i} |\alpha A_{ij}| < d_i$, because $\alpha = 0.9$, $|A_{ij}| \leq 1$. So we have

$$\text{For all } i, \quad |N_{ii}| - \sum_{j=1, j\neq i}^{n} |N_{ij}| = d_i - \sum_{j=1, j\neq i}^{n} |\alpha A_{ij}| > 0.$$

Therefore we have shown that matrix N is a strictly diagonally dominant matrix.

For a symmetric, strictly diagonally dominant matrix with all positive diagonal elements, we can use the Gershgorin Disk Theorem to prove such a matrix is positive definite. First we state the theorem:

Theorem. *(Gershgorin Disk Theorem) Let A be any $n \times n$ matrix and λ be any eigenvalue of A. Then for some i, $1 \leq i \leq n$,*

$$|\lambda - A_{ii}| \leq \sum_{j=1, j\neq i}^{n} |A_{ij}|.$$

Applying the Gershgorin Disk Theorem to matrix N, we have the bounds of its eigenvalues λ as

$$-\sum_{j=1, j\neq i}^{n} |N_{ij}| \leq \lambda - N_{ii} \leq \sum_{j=1, j\neq i}^{n} |N_{ij}|.$$

Since we have shown above that N is *strictly diagonally dominant*, we conclude

$$\lambda \geq N_{ii} - \sum_{j=1, j\neq i}^{n} |N_{ij}| > 0,$$

which means all eigenvalues of N are positive. By the Principal Axes Theorem for the symmetric matrices, we can easily show that any symmetric matrix with all positive eigenvalues is positive definite. So we conclude that N is symmetric positive definite (SPD).

Task 4

The sparse LU decomposition of M in MATLAB gives us L, U and P and Q such that

$$PMQ = LU, \quad \text{so } M = P^{-1}LUQ^{-1},$$

therefore

$$Mu = P^{-1}LUQ^{-1}u = e_1,$$

and finally

$$LU(Q^{-1}u) = Pe_1.$$

We solve the above equation in two stages:

$$Lz = Pe_1, \quad z = L\backslash(P * e_1), \quad \text{(the forward substitutiion stage)}$$

$$U(Q^{-1}u) = z, \quad Q^{-1}u = U\backslash z. \quad \text{(the backward substitution stage)}$$

Putting them together, we compute $Mu = e_1$ as follows:

$$u = Q * (U\backslash z) = Q * (U\backslash(L\backslash(P * e1)))$$

In MATLAB, the use of sparse LU decomposition of matrix M therefore can be coded as follows:

```
SpI = speye(n); % SpI is the sparse identity matrix I (n x n)
e{1}=SpI(:,1);
[P, Q, L, U] = lu(M);
u=Q*(U\(L\(P*e{1})));
```

Task 5

The main focus of this project is to use the sparse direct solver to obtain the maximization of the following function:

$$f(a, b) = \boldsymbol{e_t}^T \hat{M}(a, b)^{-1}\boldsymbol{e_s}, \tag{1}$$

where s and t are the starting node and target node, respectively, $\boldsymbol{e_s}$ and $\boldsymbol{e_t}$ are the respective standard basis vectors for columns s and t, and $\hat{M}(a, b)$ is a rank-2 update to matrix M after removing edge (a, b) from the adjacency matrix A, i.e., by setting $A_{ab} = A_{ba} = 0$ in A.

Since the size of matrix M is expected to be very large ($> 10^6$), an efficient alogrithm is a must for calculating $\hat{M}^{-1}$ without having to form, factor, and solve a linear system each time we have a new closure (closing a different edge (a, b) in the graph).

When an edge (a, b) is removed, the only change is to set $A_{ab} = A_{ba} = 0$, all other elements in A are not affected. Since

$$D = \mathrm{diag}(d_1, d_2, \ldots, d_n), \quad T = AD^{-1}, \quad \text{and} \quad M = I - \alpha T,$$

matrix $\hat{M}(a, b)$, the update to M associated with removing a single edge (a, b) in A, can be shown to be a rank-2 update (also called a rank-2 perturbation) by expressing $\hat{M}(a, b)$ as

$$\hat{M}(a, b) = M + UV, \tag{2}$$

in which the update due to closure of (a, b) is represented by the product of U and V: matrix U is $n \times 2$, and V is $2 \times n$:

$$U = \begin{pmatrix} \boldsymbol{u_1} & \boldsymbol{u_2} \end{pmatrix}, \quad V = \begin{pmatrix} \boldsymbol{e_a}^T \\ \\ \boldsymbol{e_b}^T \end{pmatrix}. \tag{3}$$

Column vectors of matrix U contain the elements which

- cancel out the elements M_{ba} and M_{ab};
- rescale all the non-zero elements in columns a and b to reflect the reduction of degrees by 1 at nodes a and b due to the closure of this edge.

By adding UV to M, we can achieve the correct update matrix $\hat{M}(a, b)$ (by proper changes of columns a and b in M):

$$\hat{M}(a, b) = M + UV.$$

Now we show the detail of the construction of U. In the following expressions, d_a and d_b are the respective column sums of $A(:, a)$ and $A(:, b)$. [1] Column vector $\boldsymbol{u_1}$ contains the update to column a of matrix M:

$$\boldsymbol{u_1} = \underbrace{\frac{\alpha}{d_a} A(:, a)}_{\text{will render } M(:,a)=0,} - \underbrace{[A(:, a) - \boldsymbol{e_b}] * \frac{\alpha}{(d_a - 1)}}_{\text{re-scale column } M(:,a) \text{, except } M_{aa}.}, \quad \text{if } d_a > 1; \tag{4}$$

$$\boldsymbol{u_1} = \frac{\alpha}{d_a} A(:, a), \quad \text{if } d_a = 1. \tag{5}$$

The first term on the right-hand side of Eq.(4) is to render $M(:, a) = 0$; the second term is to re-scale every non-zero term in $M(:, a)$ by $(d_a - 1)$, which is the new degree of node a after the closure of edge (a, b). The re-scaling does not apply to

[1] $A(:, a)$ is the MATLAB notation of column a in matrix A.

the diagonal term M_{aa}, which is 1. Similarly, column $\boldsymbol{u_2}$ can be constructed for updating column b of M:

$$\boldsymbol{u_2} = \underbrace{\frac{\alpha}{d_b}A(:,b)}_{\text{will render } M(:,b)=0,} - \underbrace{[A(:,b) - \boldsymbol{e_a}] * \frac{\alpha}{(d_b - 1)}}_{\text{re-scale column } M(:,b) \text{ except } M_{bb}.}, \quad \text{if } d_b > 1; \tag{6}$$

$$\boldsymbol{u_2} = \frac{\alpha}{d_b}A(:,b), \quad \text{if } d_b = 1. \tag{7}$$

Because $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ are two linearly independent vectors (they are not multiples of each other), their product UV is a rank-2 matrix of size $n \times n$. Therefore $\hat{M}(a,b) = M + UV$ is called a rank-2 update to matrix M.

Task 6

Now we are ready to utilize the *Sherman-Morrison-Woodbury* formula to find the inverse of a low-rank perturbation of matrix M, namely, $\hat{M}^{-1}(a,b) = (M + UV)^{-1}$, at a lower cost. The formula is

$$(M + UV)^{-1} = M^{-1} - M^{-1}U(I_{p\times p} + VM^{-1}U)^{-1}VM^{-1}. \tag{8}$$

In Eq.(8), the LU decomposition of M is already available. (Alternatively, if we are using N, Cholesky decomposition is available.) The sizes of M, U and V are $n \times n$, $n \times p$, and $p \times n$, respectively.

For the rank-2 update in this project, $p = 2$, so computing $(I_{2\times 2} + VM^{-1}U)^{-1}$ only requires a 2×2 linear system solve.

The implementation of a function is outlined below: We solve the equation of the rank-2 update to M as follows:

$$\boldsymbol{x} = (M + UV)^{-1}b = \left[M^{-1} - M^{-1}U(I_{p\times p} + VM^{-1}U)^{-1}VM^{-1}\right]\boldsymbol{b},$$

where M, U, V and b are the input matrices and vector, and $\boldsymbol{x}$ is the output vector. For Eq.(1) of this project, $p = 2$, and $\boldsymbol{b} = \boldsymbol{e_s}$, one of the standard basis vectors.

0. Perform the sparse LU decomposition of M by MATLAB, so

$$PMQ = \tilde{L}\tilde{U},$$

and store the matrices P, Q, $\tilde{L}$ and $\tilde{U}$. This should be done before calling the Sherman-Morrison-Woodbury routine if many rank-2 updates are done to the same matrix M.

1. Solve $M\boldsymbol{y} = \boldsymbol{b}$. The available LU decomposition of M is $PMQ = \tilde{L}\tilde{U}$, so $\boldsymbol{y}$ is obtained less expensively (without re-factoring M):

$$\boldsymbol{y} = Q * \left[\tilde{U}\backslash(\tilde{L}\backslash(P * \boldsymbol{b}))\right]. \tag{9}$$

   Here "\" is the MATLAB *"mldivide"* or *"backlash"* operator which solves the system of $A\boldsymbol{x} = B$.

2. It is important to recognize that we should solve $M^{-1}U$ in Eq.(8). Denoting $M^{-1}U = W$, then solve $M\boldsymbol{w}_i = \boldsymbol{u}_i$, where $\boldsymbol{w_i}$ and $\boldsymbol{u_i}$ are the ith columns of U and W, respectively. Note that W and U are of size $n \times 2$. Again, we take advantage of the availability of the sparse LU decomposition of M:

$$\boldsymbol{w_i} = Q * \left[\tilde{U}\backslash(\tilde{L}\backslash(P * \boldsymbol{u_i}))\right], \quad i = 1, 2. \tag{10}$$

3. Set up $C = I_{2\times2} + VW$ and $V\boldsymbol{y}$, then solve for $\boldsymbol{z}$ in

$$C\boldsymbol{z} = V\boldsymbol{y}. \tag{11}$$

   C is 2×2, we solve for $\boldsymbol{z}$ directly. Note that $\boldsymbol{z} = (I_{2\times2} + VM^{-1}U)^{-1}VM^{-1}\boldsymbol{b}$.

4. Finally, $\boldsymbol{x} = (M + UV)^{-1}\boldsymbol{b} = \boldsymbol{y} - W\boldsymbol{z}$.

The MATLAB code of the Sherman-Morrison-Woodbury is lited below:

```
function [x] = Sherman_Morrison_Woodbury(tL, tU, P, Q, U, V, yy)
% Given a sparse LU decomposition of matrix M
% [tL, tU, P, Q]=lu(M);
% U (n by p) and V (p by n) are the rank-p update matrices
% such that M_hat = (M + UV).
% Use Sherman-Morrison-Woodbury to find
% x=(M + PV)^{-1} b
% vector yy is the  solution  of
% M^{-1} yy = b, yy = Q*(tU\(tL\(P*b)))
% has been precomputed once and used many times.
%
% This function returns x=(M+UV)^{-1}*b
%

[n,p]=size(U);

W=Q*(tU\(tL\(P*U)));
%
%%C=I_p+V*W;
```

```
%
C=speye(p)+V*W;
%
z=C\(V*yy);
x=yy-W*z;
end
```

Task 7

(Extra credit) Find a way to bound

$$f(a,b) - e_t^T M^{-1} e_s$$

that does not require any additional linear solves after pre-processing. For example, if $t \notin \{a,b\}$, one can show

$$|f(a,b) - e_t^T M^{-1} e_s| \leq \frac{1}{d_t} \left(\frac{1+\alpha}{1-\alpha} \right) \left(\frac{w_a}{d_a - 1} + \frac{w_b}{d_b - 1} \right), \tag{12}$$

where d_k is the degree of the node k (before the update) and $w = N^{-1} e_t$.

Solution: We can estimate the bounds of $|f(a,b) - e_t^T M^{-1} e_s|$ as follows:

$$f(a,b) - e_t^T M^{-1} e_s = e_t^T \hat{M}(a,b) e_s - e_t^T M^{-1} e_s = e_t^T (\hat{M}^{-1}(a,b) - M^{-1}) e_s. \tag{13}$$

Plugging in the Sherman-Morrison-Woodbury formula for $(M + UV)^{-1}$, we obtain

$$(\hat{M}^{-1}(a,b) - M^{-1}) e_s = -M^{-1} U (I + V M^{-1} U)^{-1} V M^{-1} e_s, \tag{14}$$

therefore

$$\begin{aligned} \|e_t^T (\hat{M}^{-1}(a,b) - M^{-1}) e_s\| &= \|e_t^T M^{-1} U (I + V M^{-1} U)^{-1} V M^{-1} e_s\| \\ &\leq \|e_t^T M^{-1} U\| \cdot \|(I + V M^{-1} U)^{-1}\| \cdot \|V M^{-1} e_s\|. \end{aligned} \tag{15}$$

I cannot go further than this step because I do not know the bounds of $\|M^{-1}U\|$, etc. I may have to rely on symmbolic computation to find the bound of the norms in the equation above.

Task 8

(Extra credit) Combine the results of tasks 6-7 to find the optimal edge, neglecting any edges from consideration that change the value of f by less than a tolerance $\tau = 10^{-3}$. You may also neglect any edge involving a degree one node.

Solution: For a given set of start and target nodes (s, t), we first calculate by using either sparse LU for M, or sparse Cholesky decomposition for N, to find the value of f per s and t (the function we would like to maximize):

$$f = e_t^T M^{-1} e_s. \tag{16}$$

We then proceed to use Eq.(12) to estimate $|f(a,b) - e_t^T M^{-1} e_s|$, the "bound distribution", among all edges in the network *without* having to perform additional linear solves. Once we have obtained the upper bounds of $|f(a,b) - e_t^T M^{-1} e_s|$ for all edges, we can use Sherman-Morrison-Woodbury formula to perform an accurate calculation of f for the edge with the highest estimated bound value. The edge giving the highest value of $f(a,b)$ is the optimal edge (road) for the given (s,t).

The logic of the code is outlined as follows:

1. Read in matrix A and set up D, D^{-1}, M and N.

2. Perform LU decomposition of M. Then precompute $M^{-1}e_s$ and $N^{-1}e_t = D^{-1}M^{-1}e_t$.

3. Take the upper triangular part of A and use vectorized operation to calculate the bound estimates of each edge pair in Eq.(12).

4. Find the maximum value of the bound estimate, then use Sherman-Morrison-Woodbury routine to find $e_t^T \hat{M}(a,b) e_s$ and print out the result.

The key part in this task is to vectorize the MATLAB calculations of the upper bound in Eq.(12) for each (a,b) pair of the upper triangular part of matrix A. The difference is speed is very substantial since the number of edges is huge. We have done a large number of testing of the code called "project1_boundEst2.m". User must set s and t in the beginning of the code, and a typical output is shown below. The code listing is in the next page.

```
>> project1_boundEst2
max. bound occurs at 2, a=1, b=3

 for t=1,  s=180
the optimal edge is (1 , 3),   f=0.0702679
```

```
%CS4220 Project1: find the optimal edge of
%CA Road Network for each pair of
%given start (s) and target (t) nodes
%
%Prob=UFget(2317);
%A=Prob.A;
%
%Load the matrix from roadNet-CA.mat file
%
%
Z=load('roadNet-CA.mat');
A = Z.Problem.A;
alpha = 0.9;

%testing one (s,t) pair
%s=85688; t=85719;
s=180; t=1;

n = length(A);
d = full(sum(A))';
INZ = find(d);
n = length(INZ);
d = d(INZ); % get rid of the zero-degree nodes
A = A(INZ, INZ);
D = spdiags( d, 0, n, n );
Dinv = spdiags( 1./d, 0, n, n);
N = D -alpha*A;

T = A*Dinv;
M=speye(n) - alpha*T;
p=2; %This problem is a rank-2 update to matrix M

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

I = speye(n);
e{s}=I(:,s);
e{t}=I(:,t);
%
% since we decide to use M_hat, not N_hat, we don't
% have the need to do the Cholesky decomp for N.
% R'*R = S'*N*S, so Nw=b can be solved as
```

```
%        w = S*(R\(R'\(S'*b)));
%
%[R, pp, S]=chol(N);
%ww=S*(R\(R'\(S'*e{t})));
%ww=S*(R\(R'\(S'*e{t})));
% PMQ=tL*tU
[tL, tU, P, Q]=lu(M);
% Nw=b; w=D^{-1}( M^{-1}*b ); so ww=D^{-1}*t, where t is
% the solution based on the lu(M) decomp to solve M^{-1}b:
ww=Dinv*Q*(tU\(tL\(P*e{s})));
% yy is M^{-1} e_s we need in Sherman-Morrison-Woodbury
yy=Q*(tU\(tL\(P*e{s})));

%A is symmetric, take its upper triangular part for (a,b)
[r,c]=find(triu(A));

Nedges=length(r);
fab_bound=zeros(Nedges,1);
temp1=1/d(t)*(1+alpha)/(1-alpha);
%deg_two = find(d>1);
%Nedges2=length(deg_two);
%fab_bound(deg_two)=temp1*( ww(r(deg_two))./(d(r(deg_two))-1)+...
%                     ww(c(deg_two))./(d(c(deg_two))-1) );
%
% Vectorize MATLAB calculations of the bounds here:
%
ind=(1:Nedges)';
fab_bound(ind)=temp1*( ww(r(ind))./(d(r(ind))-1)+...
                   ww(c(ind))./(d(c(ind))-1) );

% It is  critically  important to get  rid  of  Inf  and NaN from
% the fab_bound (since we have not  filtered  out  single  degree nodes:

fab_bound( isinf(fab_bound) | isnan(fab_bound) )=0;

[opt, indsort] = sort( fab_bound, 'descend' );
[optedge, indexx] = max( fab_bound );
a=r(indsort(1)); % a, b are the optimal edge pair
b=c(indsort(1)); %
disp(sprintf('max. bound occurs at %d, a=%d, b=%d', indsort(1), a, b))
%disp(sprintf('\n (a=%d, b=%d)', a, b))
```

```
% e{a}, e{b}, u{1} etc. are cell arrays, which
% can be used for arrays of vectors or matrices.
% This is equivalent to array of any data structure
% in Java, C++ and C.

e{a}=I(:,a);
e{b}=I(:,b);

if ( d(a) > 1 )
    u{1}=alpha/d(a)*A(:,a) - (A(:,a) - e{b})*alpha/(d(a)-1);
else
    u{1}=alpha/d(a)*A(:,a);
end

if ( d(b) > 1 )
    u{2}=alpha/d(b)*A(:,b) - (A(:,b) - e{a})*alpha/(d(b)-1);
else
    u{2}=alpha/d(b)*A(:,b);
end

U = [u{1} u{2}];
V = [e{a} e{b}];  % V' is the 2xn matrix
%
% We need not generate M_hat in this code. Sherman-Morrison-
% Woodbury produces M_hat^{-1} for us.
%
% M_hat = M+U*V';
% (M + U*V')^{-1} is obtained by the following:
%
% PMQ=tL*tU
%[tL, tU, P, Q]=lu(M);
%yy=Q*(tU\(tL\(P*e{s})));

x=Sherman_Morrison_Woodbury(tL, tU, P, Q, U, V', yy);
f=e{t}'*x;

fval=full(f);
disp(sprintf('\n for t=%d,  s=%d ', t, s));
disp(sprintf('the optimal edge is (%d , %d),   f=%g', a, b, fval));
```