Proj 1: All Roads Lead to Fresno

There are many problems that involve optimizing some objective function by making local adjustments to a structure or graph. For example:

- If we want to reinforce a truss with a limited budget, where should we add new beams (or strengthen old ones)?
- After a failure in the power grid, how should lines be either taken out of service or put in service to ensure no other lines are overloaded?
- In a road network, how will road closures or rate-limiting of on-ramps affect congestion (for better or worse)?
- In a social network, which edges are most critical to spreading information or influence to a target audience?

In many cases, translating the original physical problem into a mathematical model is itself a difficult challenge. Rather than working with a "real-world" model, we are going to focus in this project on a toy model that has many real-world characteristics. Of course, in the process we also want to exercise your knowledge of linear systems, norms, and the like!

Logistics

You are encouraged to work in pairs on this project. You should produce short report addressing the analysis tasks, and a few short codes that address the computational tasks. You may use any MATLAB functions you would want.

You should be able to convince both me and your fellow students that your code is right. A good way to do this is to test thoroughly. Check residuals, compare cheaper or more expensive ways of computing the same thing, and generally use the computer to make sure you don't commit silly errors in algebra or coding. You will also want to make sure that you satisfy the efficiency constraints stated in the tasks.

You will have ten days to work on the project, at which time you should submit your report and code. This will be reviewed by two other students, following guidelines that I will distribute, after which you will have the chance to change things.

The Story

A driver in California leaves his home and promptly gets lost. The driver wants to reach some target (e.g. Fresno), but in fact ends up wandering at random until he is exhausted. Our goal is to close one road to maximally improve the expected time the driver sill spend at his target.

Mathematically, we model the road network topology by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, where n is the number of intersections or destinations and

$$A_{ij} = \begin{cases} 1, & \text{if a road connects } i \text{ to } j \\ 0, & \text{otherwise.} \end{cases}$$

The driver treats all roads as equally plausible, and so we have

$$T_{ij} = A_{ij}/d_j$$

as the probability that a driver at node j will choose to take the rode to node i, where d is the degree of node j.

Our driver follows one road per step. At each step he keeps going with probability $\alpha = 0.9$, and gives up (or runs out of gas) with probability $1 - \alpha$. Thus, if the driver starts at node s, the expected number of times we expect he will visit a target node t is

$$f = \sum_{k=0}^{\infty} (\alpha T^k)_{ts} = \left[(I - \alpha T)^{-1} \right]_{ts}.$$

Our goal is to find one road (a, b) whose closure most increases f. That is, we want to change the A_{ab} from a one to a zero in order to increase f. Moreover, we want to do this fast.

Getting Started

We will be using the California road network data from the SNAP data set; you can retrieve this as a MATLAB M-file from

http://www.cise.ufl.edu/research/sparse/matrices/SNAP/roadNet-CA.html

This is a big enough network that you will *not* want to form it, or its inverse, as a dense matrix. On the other hand, because it is a moderate-sized planar graph, sparse Gaussian elimination on $I - \alpha T$ will work fine.

Once you have downloaded the data set, you can use the following code to form the matrix $M = I - \alpha T$:

```
alpha = 0.9;
n = length(A);
d = full(sum(A))';
T = A * spdiags(1./d, 0, n, n);
M = speye(n) - alpha*T;
```

Task 1 Solve the linear system $Mu = e_1$ using MATLAB's sparse backslash operator, where e_1 is the first column of the identity matrix. Using the `tic` and `toc` commands, approximately how long does this computation take?

Task 2 Look at the documentation for the MATLAB `lu` command, and figure out how to do an LU decomposition of M with column pivoting (for sparsity) as well as row pivoting. That is, you will compute a decomposition

$$PMQ = LU.$$

For this problem, you should find that $PQ = I$; verify that this is true. Can you figure out why?

Task 3 We can write the matrix M as $M = ND^{-1}$ where $N = D - \alpha A$ and D is a diagonal matrix of degrees. Argue that in this case, N is symmetric and positive definite for $\alpha < 1$.

Task 4 Given either the sparse Cholesky (with variable reordering) or sparse LU (with column pivoting), write a short MATLAB expression to solve $Mu = e_1$ without re-factoring M (which is what happens if you use the backslash operator on M directly).

The Optimization

Your goal is to maximize

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

where $\hat{M}(a, b)$ is the matrix associated with removing the edge (a, b). You do *not* want to have to form, factor, and solve a linear system involving $\hat{M}(a, b)$ for every edge (a, b) in the network; you'll want to be smarter than that. We will use two tricks: low rank structure and norm bounds.

Task 5 Show how to write $\hat{M}(a, b)$ as a simple rank-2 update to M. Alternately, you may do the same thing for the matrix $\hat{N}(a, b)$ and N (where $M = ND^{-1}$ as above).

Task 6 The *Sherman-Morrison-Woodbury* formula says that if B, C, and $B + UCV$ are all invertible, then

$$(B + UCV^T)^{-1} = B^{-1} - B^{-1}U(C^{-1} + V^T B^{-1}U)^{-1}V^T B^{-1}.$$

Write a routine using this formula to accelerate the computation of $f(a, b)$. In addition to taking advantage of the LU factorization of M (or the Cholesky factorization of N), you should try to precompute as much as possible so that you don't have to solve too many linear systems.

Task 7 For any consistent norm, if $\|B\| < 1$ then

$$\|(I + B)^{-1}\| \leq \frac{1}{1 - \|B\|}.$$

This is sometimes called a Neumman series bound or geometric series bound. You do not have to prove it. In particular, this means that

$$\|(I - \alpha T)\|_1^{-1} \leq \frac{1}{1 - \alpha},$$

since the column sums of T are all normalized to 1.

In your Sherman-Morrison-Woodbury formula, there should be one place where you cannot precompute the solution to a linear system involving the matrix M or N. Use this bound, along with what you know more generally about norm bounds, to compute an upper bound on the effect of changing an edge in the matrix. After a little precomputation good for all edges, you should be able to evaluate your bound with *no* extra linear solves.

Task 8 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 $\tau = 10^{-3}$.