# Proj 1: Hotel California

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?

For our project, a driver in California leaves his home and wanders the roads randomly in the hope of reaching a destination. If he arrives, he never leaves[1]. At any given step, there is also a small probability that the driver will get bored and give up. Our goal in this project is to compute the mean time the driver spends on the road as a function of source and destination. Once we have determined this, we will try to figure out how to close one road to best improve this time. 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 or Python functions you might want.

Most of the code in this project will be short, but that does not make it easy. You should be able to convince both me and your partner 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.

[1]Though, according to the night man, he can check out any time he likes.

# Background

## Markov Chain Basics

A Markov chain is one of the simplest and most important types of random process. In the discrete space and time setting, a Markov chain involves a sequence of states (we label the set of all states as $\{1, \dots, n\}$) in which the state $X_{k+1}$ at time $k+1$ depends on the state at time $k$, but not on the states at any previous steps. A Markov chain is characterized by the *transition matrix* $P$ with entries

$$p_{ij} = \Pr\{X_{k+1} = j | X_k = i\}$$

Let $w^{(k)} \in \mathbb{R}^n$ be a row vector whose $n$ entries represent the probability that the chain is in each of the $n$ possible states at time $k$. In terms of the vectors $w$ and the matrix $P$, we can rewrite the identity

$$\Pr\{X_{k+1} = j\} = \sum_{i=1}^{n} \Pr\{X_k = i\} \Pr\{X_{k+1} = j | X_k = i\}$$

as the matrix iteration

$$w^{(k+1)} = w^{(k)} P = w^{(0)} P^{k+1}.$$

We write the probabilities as row vectors to be consistent with the usage in most of the literature on probability and statistics, and so that $p_{ij}$ can be read as "probability of transitioning from $i$ to $j$."

A Markov chain is *ergodic* if there is a path from every state to every other state. A Markov chain is *irreducible* if there is some number of steps $k$ such that any state can reach any others state in exactly $k$ steps with nonzero probability — in matrix terms, some power of $P$ has all nonzero elements. A Markov chain that is ergodic and regular always converges eventually to a *stationary distribution*; that is, for any starting distribution vector $w^{(0)}$,

$$w^{(k)} = w^{(0)} P^k \to w^{(*)} \quad \text{as} \quad k \to \infty.$$

The vector $w^{(*)}$ is the unique row eigenvector of $P$ associated with the eigenvalue 1. We will talk more of this eigenvector — and the other eigenvalues and eigenvectors of Markov chains — when we get to the part of the class dealing with eigenvalue problems.

## Absorbing Markov Chains

An *absorbing Markov chain* is a Markov chain with two types of states: ordinary *transient states* that the chain can move between; and *absorbing states* that the chain can enter, but never escape. Absorbing Markov chains play an important role in the analysis of random processes with an end state or goal state; examples include the classic "gambler's ruin" problem, the drunkards walk problem, or the analysis of play time for Snakes and Ladders

If we order the transient states first, the transition matrix takes the form

$$P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix}.$$

The expected number of steps in an absorbing Markov chain starting at a source node $s$ is the sum of the probability that the chain remains in a transient state at times $0, 1, 2, \ldots$, i.e.:

$$t_s = \sum_{k=0}^{\infty} \Pr\{X_k \text{ is transient} | X_0 = s\} = \sum_{k=0}^{\infty} e_s^T Q^k e = e_s^T (I - Q)^{-1} e,$$

where $e_s$ is column $s$ of an appropriately-sized identity matrix and $e$ is the vector of all ones. We will assume that there is a path from every transient state to some absorbing state, which is necessary and sufficient to guarantee that the expected number of steps is finite.

One way to get an absorbing Markov chain is to take an ordinary Markov chain and replace some states with an absorbing state; that is, $P$ might be a modified version of the transition matrix

$$\tilde{P} = \begin{bmatrix} Q & R \\ X & Y \end{bmatrix}$$

in which we've replaced the second set of states with absorbing states. We can compute the vector $t = (I - Q)^{-1} e$ representing the expected number of steps in the absorbing case in terms of an extended system involving $\tilde{P}$:

$$\begin{bmatrix} I - Q & -R & 0 \\ -X & I - Y & I \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} t \\ v \\ \lambda \end{bmatrix} = \begin{bmatrix} e \\ 0 \\ 0 \end{bmatrix}. \tag{1}$$

**Task 1** Show that the vector $t$ from the extended linear system (1) is the same as the vector $t = (I - Q)^{-1} e$.

## Mean Hitting Times

In an ergodic chain, the *mean hitting time* is the expected number of steps to first reach some target state or states from a given starting distribution. To compute mean hitting time $m_{\sigma\tau}$ to get from some start state $\sigma$ to a target state $\tau$, we can think of modifying the original Markov chain so that the target state is absorbing. In terms of the extended system formulation (1) in the last section, we have $m_{\sigma\tau} = t_\sigma$ where $t$ is given by

$$\begin{bmatrix} I-P & e_\tau \\ e_\tau^T & 0 \end{bmatrix} \begin{bmatrix} t \\ \lambda \end{bmatrix} = \begin{bmatrix} e \\ 0 \end{bmatrix},$$

or, more concisely

$$m_{\sigma\tau} = \begin{bmatrix} e_\sigma \\ 0 \end{bmatrix}^T \begin{bmatrix} I-P & e_\tau \\ e_\tau^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} e \\ 0 \end{bmatrix}.$$

If you read on your own about mean hitting times, you are likely to see a different formula involving the *fundamental matrix* $Z = (I - P + W)^{-1}$ where $W = \lim_{k\to\infty} P^k = ew^T$. In terms of $Z$, we have

$$m_{\sigma\tau} = \frac{z_{\tau\tau} - z_{\sigma\tau}}{w_\tau}.$$

It is easy enough to test by experiment that these formulations give the same result, but we will stick to the bordered system formulation for this project.

We will also consider a *damped hitting time*[2]. This corresponds to the expected number of steps in a Markov chain that ends when we reach a given target or when we give up (which happens with probability $1-\eta$ from each state). In terms of the extended system formulation, this gives

$$\begin{bmatrix} I-\eta P & e_\tau \\ e_\tau^T & 0 \end{bmatrix} \begin{bmatrix} t \\ \lambda \end{bmatrix} = \begin{bmatrix} e \\ 0 \end{bmatrix}.$$

For $\eta < 1$, the $(1,1)$ block of this bordered matrix becomes nonsingular, which makes it simpler to play with block Gaussian elimination.

**Task 2** Use Gaussian elimination and the fact that $Pe = e$ to show that the damped hitting time from a start state $\sigma$ to a target $\tau$ is

$$m_{\sigma\tau} = \frac{1}{1-\eta}\left(1 - \frac{e_\sigma^T (I-\eta P)^{-1} e_\tau}{e_\tau^T (I-\eta P)^{-1} e_\tau}\right).$$

[2]Unlike other vocabulary in this section, “damped hitting time” is not a standard term

### The Road Network

We represent the road network 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

$$P = D^{-1}A$$

where $D$ is a diagonal matrix of node degrees ($d_j$ is the degree of node $j$).

We will use 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 $I - \eta P$ will work fine. Once you have downloaded the data set, use `problem_load.m` to form the matrix $A$.

## Getting Started

As a test case, you might want to consider trying to get your random walker to start at node $\sigma = 1$ and arrive at node $\tau = 180$. Use the damping parameter $\eta = 0.9$. We will denote by $F$ the matrix $I - \eta P$.

**Task 3** Solve the linear system $Fu = e_\tau$ using MATLAB's sparse backslash operator, and use the formula from Task 2 to compute the damped hitting time. In Python, you can use the `spsolve` or `factorized` methods from `scipy.sparse.linalg`. According to the `tic` and `toc` commands, approximately how long does this computation take?

**Task 4** We can write the $F = D^{-1}H$ where $H = D - \eta A$ is symmetric and positive definite. Show that for this case, (1) is equivalent to

$$m_{\sigma\tau} = \frac{1}{1-\eta}\left(1 - \frac{e_\sigma^T H^{-1} e_\tau}{e_\tau^T H^{-1} e_\tau}\right).$$

Look at the documentation for the MATLAB `chol` command, and figure out how to do a sparse Cholesky decomposition of $H$ with column pivoting (for sparsity). That is, you will compute a decomposition

$$Q^T H Q = LL^T.$$

Given the pivoted Cholesky factorization of $H$, show how to compute $m_{\sigma\tau}$ for different $\sigma$ and $\tau$ without re-factoring $H$ (which is what happens if you use the backslash operator on $F$ directly).

## The Optimization

Your goal is to minimize the mean hitting time on removing an edge $(a, b)$ from the network, which we will denote as $\hat{m}_{\sigma\tau}(a, b)$ (or just $\hat{m}_{\sigma\tau}$). You do *not* want to have to form, factor, and solve a new linear system 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.

### Low Rank Updates

To exploit low rank structure, we need the *Sherman-Morrison-Woodbury* formula says that if $B$, $C$, and $B + UCV$ are all invertible, then

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

The formula can easily by showing that if $(B + UCV^T)x = b$ then we also satisfy the extended system

$$\begin{bmatrix} B & U \\ V^T & C^{-1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

I have always found the extended system formulation more memorable and algebraically tractable than the original formula.

**Task 5** Let $\hat{H}(a, b)$ denote the $H$ matrix after removing an edge $(a, b)$. Show how to write $\hat{H}(a, b)$ as a simple rank-2 update to $H$.

**Task 6** Write a routine using Sherman-Morrison-Woodbury (or an equivalent extended system) to accelerate the computation of $\hat{m}_{\sigma\tau}(a, b)$. In addition to taking advantage of the previous Cholesky factorization of $H$ (and no other factorizations should be required), you should try to precompute as much as possible so that you don't have to solve too many linear systems.

## Norm bounds

The update formula gives us a means to find the effect of removing an edge without re-factoring the matrix. But there are enough edges that even doing one or two linear solves for each edge becomes expensive. Fortunately, most edges intuitively have little change on a given damped hitting time. We can formalize this as a bound that lets us "rule out" uninteresting edges without solving any linear systems at all. In particular, we can show that if $H = D - \eta A$ and $\hat{H} = \hat{D} - \eta \hat{A}$ are matrices associated with the original graph and the graph with edge $(a, b)$ removed (assuming $a$ and $b$ both have degree greater than one), then

$$\|\hat{x} - x\|_\infty \leq \frac{1+\eta}{1-\eta} \frac{\max(x_a, x_b)}{\min(d_a - 1, d_b - 1)}. \tag{2}$$

From this, we can obtain informative upper and lower bounds on $\hat{m}_{\sigma\tau}(a, b) - m_{\sigma\tau}$ to guide an optimization.

**Task 7** Let $\hat{H} = H - UCU^T$ be the modified version of $H$ after removing an edge $(a, b)$ from the graph, where $U = \begin{bmatrix} e_a & e_b \end{bmatrix}$. Let $\hat{H}\hat{x} = e_\tau$ and $Hx = e_\tau$ be solutions of the linear system before and after modification. Show that

$$x - \hat{x} = \hat{H}^{-1}UCU^T x = \hat{F}^{-1}\hat{D}^{-1}UCU^T x,$$

and use this decomposition and norm bounds to show (2).
*Hint*: Use $\|\hat{P}\|_\infty = 1$ and a Neumann bound to control $\|\hat{F}^{-1}\|_\infty$.

**Task 8** From (2), show how to compute bounds on

$$\hat{m}_{\sigma\tau}(a, b) - m_{\sigma\tau}$$

that do not require any additional linear solves after the computations that go into computing $m_{\sigma\tau}$.

## Intelligent ordering

The bounds developed in Task 8 should be enough to show with little computation that for most edges, $\hat{m}_{\sigma\tau}(a, b)$ differs negligibly from $m_{\sigma\tau}$. We will ignore all edges that change $m_{\sigma\tau}$ by less than a tolerance of $\tau = 10^{-2}$.

**Task 9** Write a fast code to apply the bounds in Task 8 to find the road whose removal minimizes $\hat{m}_{\sigma\tau}(a, b)$. You should only consider edges that reduce $m_{\sigma\tau}$ by at least $10^{-2}$; if there are no such edges in the graph, report a diagnostic. I recommend checking potential edges in ascending order by the lower bound from Task 8; you can quit as soon as the lower bound on all remaining edges is greater than the minimum value of $\hat{m}_{\sigma\tau}(a, b)$ for all the edges that have been explicitly checked.

**Task 10** Test for the parameters mentioned before: $\sigma = 1$, $\tau = 180$, and $\eta = 0.9$. How long does it take you to do the optimization? How long the optimization would take without the filtering and ordering tricks? How long would it take if we knew nothing about low rank updates and had to re-factor to test each edge?

## Afternotes

1. Almost all the tasks in this project boil down to block factorization, using sparse factorizations, and norm bounds. Nothing should take many lines of code if you do it right. Nonetheless, the project is not trivial — so ask questions! Office hours and Piazza are your friends.

2. If you follow the intended path, none of the computations should be too expensive. However, the road network is large enough that you will cause yourself serious pain if you attempt to form it as a dense matrix. You may also run into trouble if you attempt a direct factorization without permuting for sparsity.

3. The case $\eta = 1$ (i.e. no damping) is harder in many ways: the matrix $I - P$ is singular, and the bounds that we derived along the way fall apart. I'd be willing to give significant extra credit if anyone can figure out a similar optimization in this case.