5.4 Random walks on directed graphs

An important application of random walks on directed graphs comes from trying to establish the importance of pages on the World Wide Web. One way to do this would be to take a random walk on the web and rank pages according to their stationary probability. However, several situations occur in random walks on directed graphs that did not arise with undirected graphs. One difficulty occurs if there is a node with no out edges. When the walk encounters this node the walk disappears. As a consequence the probabilities no longer sum to one. Another difficulty is that a node with no in edges is never reached. One way to resolve these difficulties is to introduce a random restart condition. At each step, with some probability *r* jump to a node selected uniformly at random and with probability $1-r$ select an edge at random and follow it. If a node has no out edges, the value of *r* for that node is set to one. This has the effect of converting the graph to a strongly connected graph. Assuming the graph is not periodic, this insures that the stationary probability exists.

Theorem 5.XXX: A random walk on a finite directed graph consisting of a single strongly connected component that is not periodic has the following properties:

(1) There is a unique stationary probability distribution π which is the first singular vector of the adjacency matrix.

(2) The number of times $N(i,t)$ that the Markov process visits state *i* in *t* steps satisfies $\lim_{t\to\infty} \frac{N(i,t)}{t} = \pi_i$

(3) Each state will be visited with expected time $\frac{1}{\pi_i}$

■

Finite Markov processes

A *Markov process* is a random process in which the probability distribution for the future behavior depends only on the current state, not on how the process arrived at the current state. Markov processes are equivalent mathematically to random walks on directed graphs but the literature on the two topics developed separately with different terminology. Since much of the terminology of Markov processes appears in the literature on random walks, we introduce the terminology here to acquaint the reader with it.

A state in a Markov process is *persistent* if it has the property that should the state ever be reached, the random process will return to it with probability one. This means that the state is in a strongly connected component with no out edges. Consider the directed graph with three strongly connected components A, B, and C. Starting from any

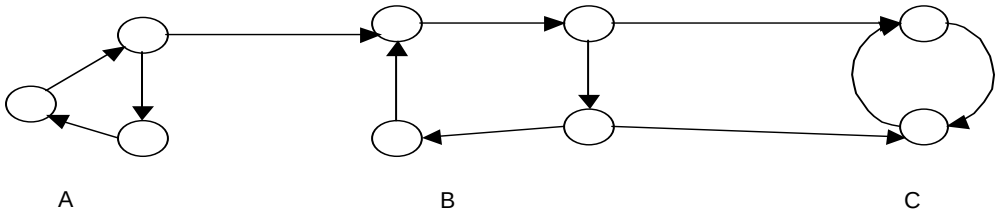

Figure 5. Vertices in C are persistent since if they are ever reached they will be reached again with probability one.

vertex in A there is a nonzero probability of eventually reaching any vertex in A. However, the probability of returning to a vertex in A is less than one and thus vertices in A and similarly vertices in B are not persistent. From any vertex in C, the walk will return with probability one to that vertex eventually since there is no way of leaving the component C. Thus, vertices in C are persistent.

A state is *periodic* if it is contained only in cycles in which the gcd of the cycle lengths is greater than one. A Markov process is *irreducible* if it consists of a single strongly connected component. An *ergodic* state is one that is aperiodic and persistent. A Markov chain is *ergodic* if all states are ergodic. In graph theory this corresponds to a single strongly connected component that is aperiodic.

Page rank and discovery time

The page rank of a vertex in a directed graph is the stationary probability of the vertex. We assume some restart value, say r=0.15, is used. The restart insures that the graph is strongly connected. The page rank of a page is the fractional frequency with which the page will be visited over a long period of time. If the page rank is *p*, then the expected time between visits or return time is 1/p. Notice that one can increase the pagerank of a page by reducing the return time and this can be done by creating short cycles.

Consider a vertex that has a single edge in and a single edge out. For simplicity normalize the frequency of walks through the edge in to be one. By conservation of the walker, the out edge must also have value one and the page rank of the vertex is one. If one adds a self loop the pagerank goes up to two, and adding *k* self loops increases the page rank to *k*+1. To see this let *x* be the number of walks entering the node. Then $\frac{k}{k+1}x$ walks leave the node by the self loops and return. Thus $x = \frac{k}{k+1}x + 1$ or $x = k + 1$. Of course, this would reduce the frequency of walks in, but let's ignore that for a moment. What prevents one from increasing the page rank of a page arbitrarily? The answer is the restart. We neglected the 0.15 percent that is taken off for the random restart. With this factor taken into account the equation for x is $x = \frac{0.85k}{k+1}x + 1$. Adding a single loop only increases the page rank to 1.74. In the limit as *k* goes to infinity, adding *k* loops increases the page rank to 6.67. However, search engines ignor self loops when calculating page rank so one would need to add a vertex and create a loop of length two.

Hitting time

Related to page rank is another quantity called hitting time. Hitting time is closely related to return time and thus to the reciprocal of page rank. One way to return to a vertex *v* is by a path in the graph from *v* back to *v*. Another way is to start on a path that encounters a restart followed by a path from the random restart vertex to *v*. The time to reach *v* after a restart is the hitting time. Thus, return time is clearly less than the expected time until a restart plus hitting time. The fastest one could return would be if there was a short loop of length two since self loops are ignored in calculating page rank. If *r* is the restart value then the loop would be traversed with at most probability $(1-r)^2$. With probability $r+(1-r)r=(2-r)r$ one restarts and then hits *v*. Thus, the return time is at least $(1-r)^2+(2-r)r(\text{hitting time})$. Combining these two bounds yields

$$(1-r)^2+(2-r)rE(\text{hitting time}) \le E(\text{return time}) \le E(\text{hitting time})$$

The relationship between return time and hitting time can be used to see if a vertex has unusually high probability of short loops. However, there is no efficient way to compute hitting time for all vertices as there is for return time. For a single vertex *v* one can compute hitting time by removing the edges out of the vertex *v* for which one is computing hitting time and then run the page rank algorithm for the new graph. The hitting time for *v* is the reciprocal of the page rank in the graph with the edges out of *v* removed. Since computing hitting time for each vertex requires removal of a different set of edges, the algorithm only gives the hitting time for one vertex at a time. Since one is probably only interested in the hitting time of vertices with low hitting time, an alternative would be to use a Monte Carlo procedure.

Spam

Suppose one has a web page and would like to increase its page rank by creating some other web pages with pointers to the original page. The abstract problem is the following. We are given a directed graph *G* and a vertex *v* whose page rank we want to increase. We may add new vertices to the graph and add edges from *v* or from the new vertices to any vertices we want. We cannot add edges out of other vertices. We can also delete edges from *v*.

Page rank is the stationary probability for vertex *v* with random restarts. If we delete all existing edges out of *v*, create a new vertex *u* and edges $v \to u$ and $u \to v$, then the page rank will be increased since any time the random walk reaches *v* it will be captured in the loop $v \to u \to v$. A search engine can counter this strategy by more frequent random restarts.

A second method to increase page rank would be to create a star consisting of a vertex *v* at its center along with a large set of new vertices each with an edge directed to *v*. These new vertices will sometimes be chosen for the target of the random restart and hence the vertices increase the probability of random walk being at *v*. This second method is countered by reducing the frequency of random restarts.

Notice that the first technique of capturing the random walk increases page rank but not hitting time. One can negate the impact of someone capturing the random walk on page rank by increasing the frequency of random restarts. The second technique of creating a star increases page rank due to random restarts and decreases hitting time.

If one uses very few random restarts the star does not increase page rank much. One can check if the page rank is high and hitting time is low in which case the page rank is likely to have been artificially inflated by the page capturing the walk with short cycles.

Personalized page rank

In computing page rank one uses a restart, typically 0.15, in which at each step with the restart probability instead of taking a step in the graph one goes to a node selected uniformly at random. In personalized page rank instead of selecting a vertex uniformly at random one selects a vertex according to a personalized probability distribution. Often the distribution has probability one for a single vertex and whenever the walk restarts it restarts at that vertex.

Algorithm for computing page rank

Now let α be the restart probability with which the random walk jumps to an arbitrary vertex. With probability $1-\alpha$ the random walk selects a vertex uniformly at random from the set of adjacent vertices. Let *p* be a row vector denoting the page rank and let G be the adjacency matrix with rows normalized to sum to one. Then

$$p = \tfrac{\alpha}{n}(1,1,\cdots,1) + (1-\alpha)\, pG$$

or

$$p = \tfrac{\alpha}{n}(1,1,\cdots,1)\left(I - (1-\alpha)G\right)^{-1}.$$

By replacing the vector $\frac{1}{n}(1,1,\cdots,1)$ by a vector s whose elements sum to one we can compute a personalized page rank.

ADD CACULATION OF FIRST SINGULAR VECTOR FOR CALCULATING PAGE RANK

Exercises

Exercise 5.18: Consider a strongly connected directed graph. In the steady state calculate the flow through each edge of a random walk.

Exercise 5.19: Create a random directed graph with 200 nodes and roughly eight edges per node. Add a special restart node with a directed edge to every node in the graph. Add an edge from each node in the graph to the restart node but weight these edges to reflect how frequently the random walk should restart. Add *k* new nodes and calculate

the page rank with and without directed edges from the *k* added nodes to node 1. How much does adding the *k* edges change the page rank of nodes for various values of *k* and restart frequency? How much does adding a loop at node 1 change the page rank? To do the experiment carefully one needs to consider the page rank of a node to which the star is attached. If it has low page rank its page rank is likely to increase a lot.
Simplify above exercise by showing how to solve with restart with out modifying the graph.
Exercise 5.20: Repeat the above experiment for hitting time.

Exercise 5.XXX: Search engines ignor self loops in calculating page rank. Thus to increase page rank one needs to resort to loops of length two. By how much can you increase the page rank of a page by adding a number of loops of length two?

Exercise 5.21: Can one increase the page rank of a vertex *v* in a directed graph by doing something some distance from *v*? The answer is yes if there is a long narrow chain of vertices into *v* with no edges leaving the chain. What if there is no such chain?

Exercise 5.22: Given a very large directed graph with many nodes of out degree one or in degree one, can one compute page rank of a reduced graph in which the nodes of in degree or out degree one have been merged and then compute the page rank of the original graph from the pagerank of the reduced graph? Does the method work if there are random restarts?

Exercise 5.23: If we model random restarts by adding a restart vertex do we get the same results as if we eliminated the restart vertex and added n^2 edges?

Exercise 5.24: Consider modifying personal page rank as follows. Start with the uniform restart distribution and calculate the stationary probabilities. Then run the personalized page rank algorithm using the stationary distribution calculated instead of the uniform distribution. Keep repeating until the process converges. That is, we get a stationary probability distribution such that if we use the stationary probability distribution for the restart distribution we will get the stationary probability distribution back. Does this process converge? What is the resulting distribution? What distribution do we get for the graph consisting of two vertices u and v with a single edge from u to v?

Exercise 5.25: (a) What is the hitting time for a vertex in a complete directed graph with self loops?
(b) What is the hitting time for a vertex in a directed cycle with n nodes?

Exercise: Using a web browser bring up a web page and look at the source html. How would you extract the url's of all hyperlinks on the page if you were doing a crawl of the web? With Internet Explorer click on "source" under "view" to access the html representation of the web page. With Firefox click on "page source" under "view".

Exercise: Sketch an algorithm to crawl the World Wide Web. There is a time dealy between the time you seek a page and the time you get it. Thus you cannot wait until the page arrives before starting another fetch. There are conventions that must be obeyed if

one were to actually do a search. Sites specify information has to how long or which files can be searched. Do not attempt an actual search without guidance from a knowledgeable person.

References

Aldous and Fill, Reversible Markov Chains and Random Walks on Graphs
http://www.stat.berkeley.edu/~aldous/RWG/book.html