IME 625 Project
Applications of Graph Theory in Markov Chains

Abhinav Garg, 14013
Anuj Nagpal, 14116
Raushan Joshi, 14537
Rutuj Jugade, 14572

April 2017

Introduction

Markov chains are widely used in the modeling of various physical and conceptual process that evolve over time. For example, the spread of disease within a population, variations of stock market prices, the diffusion of liquids across a semi porous membrane etc. Each of these cases is characterized by the property that the system can be found in any of a finite number of states, and transitions between states occur at discrete instants according to specified probabilities.

Our objective here is to analyze discrete time markov chain (DTMC) processes using a graph-theoretic approach which provides a useful visual representation of the process.

The graph-theoretic representation immediately suggests several computational schemes for calculating important structural characteristics of the process. The properties of the corresponding Markov Chain can easily be deduced from transition matrix manipulation using appropriate data structures and algorithmic techniques.

Literature Review

Let { X_n n = 0,1,2,..., } be a stochastic process that takes on a finite or countable number of possible values. If the present state is denoted by X_n , the future state by X_{n+1} and the past states are represented by X_{n1} to X_0 , then,

P(X_{n+1} = j | X_n = i, $X_{n-1} = i_{n-1}$,..., $X_1 = i_1$, $X_0 = i_0$) = P_{ij} and

P(X_{n+1} = j | X_n = i) = P_{ij}

where P_{ij} is the transition probability from state i to state j, and the is represent the various states. This property is know as the Markov Property. Essentially, it means that the process (and hence the probabilities) of moving from one state of the system to another depends only on the current or most recent knowledge.

Implementation Details

We have developed a generalised tool that inputs a transition matrix and outputs various interesting properties of the MC.

Note :- The MC must be Stationary and number of states should be finite.

driver.m

This is the main driver function which performs calls to various other function:- To run the code we have to write transition matrix in matrix file (note:- that you must write fractions in reduced form i.e. 1/2 as 0.5). After this just run driver.m

Validation.m

This is the first function that ime calls. This function validates if the given transition matrix is valid i.e. all the row sums are one and it's a square matrix etc.

StepTransition.m

This function uses ChapmanKolmogorov equations to find the nth step transition. Input for the step size needs to be provided inside test.

Scc.m

This is the most complicated function in our code. This function is used to calculate the communicating block inside a markov chain. For this this function first computes the Strongly connected component of a directed graph. Then after this it anlysis these connected components to determine if a particular component is a communicating block i.e. all the edges comes inside it.
This function outputs group_num which is a mapping contaning to which communicating block a particular state belongs.
It also outputs group_cnt which represents number of communicating blocks.
It also outputs communicating array which contains indices of all the communicating blocks

merge.m

This function takes a markov chain and reduces all the communicating blocks into a single state. Also for each communicating block it calls Periodicity.m and StationaryDistr.m which computes the period and staationary distribution of a markov chain.

Beautify.m

After the last function we'll be left with an irreducible absorbing MC and We can then calculate it's Absorption time, Hitting Probablities etc. But to compute all these we need our transition matrix in a particular form i.e it needs matrix in form of

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

Beautify performs exactly this function and along with modified matrix it also gives dictionary to get original states from transformed states.

ExpectedNumVisits.m

This code calculates expected number to a state before absorption as W = $(\text{I-Q})^{-1}$. Where W is called fundamental matrix.

HittingProb.m

This function calculates the hitting probability as u = w*r.

Note:- That logic for above 3 functions is taken from book an introduction to stochastic process page 169-173.

Periodicity.m

For finding the periodicity ofa markov chain $\mathcal{M}$, we are using a beautiful algorithm:

- From an arbitrary root node, perform a breadth-first search of G producing the rooted tree T
- The period g is given by gcd(val(e) > 0 : e $\notin$ G/T) where val(e) = level(i) - level(j) + 1 where e is an edge from i to j in G/T.

To establish the correctness of Algorithm, it is sufficient to show that g = gcd(val(e)>0: e $\in$ T) divides the length of an arbitrary cycle W in G. If so it must divide the period d, the gcd of all cycle lengths in G. Using the proof from the paper given in references, we got that d divides g, then we must have g=d.

Relative to the breadth-first search tree T, let the edges E of G be partitioned into sets D and R. Set D consists of down edges (i, j), in which level(j)=level(i)+1, and set R contains the remaining edges. Notice that all tree edges are in D, all back edges are in R, whereas cross edges can be in either set. Also edge e has val(e)>0 precisely when e $\in$ R.Now let Q = [i_0, i_1,...,i_k] be any path of length l(Q)=k in G and define val(Q)=$\sum_{s=0}^{k1}$ val(i_s,i_{s+1}). Since val(i_s,i_{s+1})=level(i_s) level(i_{s+1}) + 1, it then follows that val(Q)= level(i_0)level(i_k)+l(Q). In particular if W is any cycle from node i to itself in G, then val(W)= level(i)-level(i)+l(W)=l(W). This gives l(W)=val(W)=(val(e):e $\in$ W) = (val(e):e $\in$ W $\cap$ R)).However, each term in the last sum is positive and divisible by g (the gcd of all positive non-tree values), so g divides l(W), as required.

StationaryDistr.m

Given the transition matrix, this code will return the stationary distribution of a positive recurrent Markov Chain by solving simultaneous system of linear equations.

Implemetation of driver.m

```
transition = dlmread('matrix',' ');
transition
%Periodicity(transition);
%StationaryDistr(transition);
if Validation(transition) == 0
    return
end
printmat(transition, 'Your Transtion Matrix', num2str(1:size(transition,1)),
    num2str(1:size(transition,1)))
fp = fopen('test','r');
format = '%s';
a = fscanf(fp, format);
a=a';
StepTransition(transition, a(1));
%%Works needs to be done here .From here onwards I have assumed absorbing
[group_num group_cnt communicating] = Scc(transition)
transition_m = merge(transition,group_num, group_cnt, communicating)
%%MC.

[p q r dir] = Beautify(transition_m);
w = ExpectedNumVisits(q,dir);
u = HittingProb(w,r,dir);
```

Implemetation of Scc.m

```
function [group_num group_cnt communicating] = Scc(transition)
    transitionR = transition';
    n = size(transition,1);
    stk = zeros(1,n+1);
    stk(1) = 1;
    visited = zeros(1,n);
    group_cnt=0;
    group_num = zeros(1,n);
    function [] = fill_forward(x)
```

```
        visited(x) = 1;
        for i = 1 :n
            if transition(x,i)~=0 && visited(i) == 0
                fill_forward(i)
            end
        end
        stk(1) = stk(1) + 1;
        stk(stk(1)) = x;
    end

    function [] = fill_backward(x)
        visited(x) = 0;
        group_num(x) = group_cnt;
        for i = 1 : n
            if transitionR(x,i)~=0 && visited(i) == 1
                fill_backward(i)
            end
        end
    end

    for i = 1:n
        if visited(i) == 0
            fill_forward(i)
        end
    end
    stk

    for i = stk(1) : -1 : 2
        if visited(stk(i)) == 1
            group_cnt = group_cnt + 1;
            fill_backward(stk(i))
        end
    end
    group_num = [group_num ; 1:n];
    [m k] = sort(group_num(1,:));
    group_num = group_num(:,k);
    j=1;
    communicating = [];
    for i = 1:group_cnt
        out_group = [];
        real_out_group = [];
        while(j<=n && group_num(1,j)==i)
            out_group = [out_group group_num(2,j)] ;
            for k = 1:n
                if transition(group_num(2,j),k) ~= 0
                    real_out_group = [real_out_group k];
                end
            end
            j = j+1;
        end
        real_out_group = [real_out_group out_group];
        if size(unique(real_out_group)) == size(out_group)
            str = sprintf('%d is a absorbing block',i)
            communicating = [communicating i];
        end
    end
end
```

Implemetation of Beautify.m

```
function [p q r dir] = Beautify(transition)
```

```
    n = size(transition,1);
    dir = 1:n;
    j=n;
    p= transition;
    i=1;
    absorbing = 0;
    while i <= j
        if p(i,i) == 1
            str = sprintf('%d is absorbing',dir(i))
            if i~=j
                p = p([1:i-1 j i+1:j-1 i j+1:n],:);
            else
                p = p([1:i-1 i i+1:n],:);
            end
            if i~=j
                p = p(:, [1:i-1 j i+1:j-1 i j+1:n]);
            else
                p = p(:, [1:i-1 i i+1:n],:);
            end
            s = dir(j);
            dir(j)=dir(i);
            dir(i)= s;
            j = j - 1;
            i = i - 1;
        end
        i =  i + 1;
    end
    str = sprintf('total number of absorbing states')
    absorbing = n-j
    q = p([1:j],[1:j])
    r = p([1:j],[j+1:end])
end
```

Implemetation of ExpectedNumVisits.m

```
function [w] = ExpectedNumVisits(q, dir)
    str = sprintf('Expected Number of visits to states before absorption')
    w = inv(eye(size(q,1))-q);
    printmat(w, 'Matrix for number of visits', num2str(dir([1:size(q,1)])),
   num2str(dir([1:size(q,1)])))
    str = sprintf('Expected Time for absorption')
    time = sum(w,2);
    printmat(time, 'Matrix for expected time', num2str(dir([1:size(q,1)])),'
   absorption')
end
```

Implemetation of HittingProb.m

```
function [u] = HittingProb(w, r, dir)
    str = sprintf('Absorption/Hitting Probablity')
    u = w*r;
    printmat(u, 'Hitting Probablity', num2str(dir([1:size(w,1)])), num2str(dir([
   size(w,1)+1:end])))
end
```

Implemetation of Periodicity.m

```
function [periodicity] = Periodicity(transition)
%       transition = [
%       0       0       1       0       0       0    0    0    0;
%       1       0       0       0       0       0    0    0    0;
%       0       0       0       0.5     0.5       0;
%       0       1       0       0       0       0;
%       0       0       0       0       0       1;
%       0       0       0       1       0       0];
%      str = sprintf('Periodicity Debug Statements')
    G = digraph(transition);
    plot(G);
    V = bfsearch(G, 1);
    T = bfsearch(G,1,'edgetonew');
    Dist = zeros(size(transition,1),1);
    Dist(1) = 0;
    for i=1:size(T,1)
        Dist(T(i,2)) = Dist(T(i,1)) + 1;
    end
    Diff = setdiff(G.Edges.EndNodes, T, 'rows');
    Val = zeros(size(Diff,1),1);
    for i=1:size(Diff,1)
        Val(i) = abs(Dist(Diff(i,1)) - Dist(Diff(i,2))) + 1;
    end
    Val
    periodicity = Val(1);
    for j=2:size(Val,1)
        periodicity = gcd(periodicity, Val(j));
    end
    periodicity
end
```

Implemetation of StationaryDistr.m

```
function [statdistr] = StationaryDistr(transition)
    %transition = [0.5 0.5; 0.4 0.6];
    transition = transition.';
    for i=1:size(transition,1)
        transition(i,i) = transition(i,i) - 1;
    end
    transition = [transition; ones(1,size(transition,1))];
    A = zeros(size(transition,1)-1,1);
    A = [A; 1];
    str = 'stationary distribution is:'
    linsolve(transition,A)
end
```

Implemetation of StepTransition.m

```
function [] = StepTransition(transition, a)
    if str2num(a) < 2
        return
    else
        str = sprintf('%d step transition matrix',str2num(a))
        transition^str2num(a)
    end
end
```

Implemetation of Validation.m

```
function [valid] = Validation(transition)
    valid = 1;
    if size(transition,1) ~= size(transition,2)
        str = sprintf('You dont have a square matrix \n terminating')
        valid = 0;
        return
    end

    for i = 1:size(transition,1)
        if sum(transition(i,:)) ~= 1
            str = sprintf('sum of row %d is not 1',i)
            valid = 0;
            return
        end
    end
end
```

Implemetation of merge.m

```
function [transition_m] = merge(transition,group_num, group_cnt, communicating)
i=1;
marker = 0
transition_m = ones(group_num(1,size(transition,1)),group_num(1,size(transition
    ,1)));
a=1;
n = size(transition,1);
node_to_grp = ones(size(transition,1));
for x = 1:n
    node_to_grp(group_num(2,x)) = group_num(1,x);
end
m = group_num(1,n);
while( a <= size(transition,2))
    arr = [group_num(2,a)]
    while a+1<=size(transition,2) && group_num(1,a)==group_num(1,a+1) && sum(
    find(group_num(1,a)==communicating))~=0
        a=a+1;
        arr = [arr group_num(2,a)];
    end
    a=a+1;
    w = sum(transition(arr,:),1);
    q = zeros(1,m);
    for x = 1:n
        q(node_to_grp(x))=w(x) + q(node_to_grp(x));
    end
    transition_m(group_num(1,a-1),:)= q;
    w = sum(transition(:,arr),2);
    q = zeros(1,m);
    for x = 1:n
        q(node_to_grp(x))=w(x) + q(node_to_grp(x));
    end
    transition_m(:,group_num(1,a-1))= q';
    if transition_m(group_num(1,a-1),group_num(1,a-1)) > 1
        transition_m(group_num(1,a-1),group_num(1,a-1)) = 1;
    end
    if size(arr,2) > 1
        %%here function by anuj will come as
        %%anuj(transition([i:marker],[i:marker]))
        Periodicity(transition(arr,arr));
        StationaryDistr(transition(arr,arr));
    end
end
end
```

References

1. Graph theory in MC paper, http://cecas.clemson.edu/ shierd/Shier/markov.pdf

2. An Introduction To Stochastic Modeling, Howard M.Taylor ; Samuel Karlin. Page:- 169-173