Math 4430 - Project
METROPOLIS ALGORITHM FOR EXPECTATION

Toan Cao - Samuel Lim - My Luu - Songge Shi

November 2020

1 Introduction

The Metropolis algorithm gives a way of producing samples from a target probability distribution μ on a discrete state space U with all $\mu_i > 0$.
It uses an irreducible transition matrix Q on U that is symmetric. Then we define new transition probabilities as follows

$$p_{ij} = \begin{cases} q_{ij} min(1, \mu_j/\mu_i) & \text{if } i \neq j \\ 1 - \sum_{k \neq i} q_{ik} min(1, \mu_k/\mu_i) & \text{if } i = j \end{cases}$$

Starting from state i, use Q to propose a state j to move to. If $\mu_j \geq \mu_i$ then we accept that move. If $\mu_j < \mu_i$ then we accept the move with probability $\dfrac{\mu_j}{\mu_i} < 1$ and reject the move otherwise.

2 Prove that P is reversible with respect to μ. Conclude that μ is an invariant probability distribution for P

We need to show that $\mu_i p_{ij} = \mu_j p_{ji}$ for all $i \neq j$

For any $\mu_i = \mu_j$:

$$\mu_i p_{ij} = \mu_i q_{ij} = \mu_j q_{ji} = \mu_j p_{ji}$$

Without loss of generality, we assume that $\mu_j > \mu_i$. Then

$$p_{ij} = q_{ij} \textbf{ and } p_{ji} = q_{ji}\frac{\mu_i}{\mu_j}$$

Therefore

$$\mu_i p_{ij} = \mu_i qij = \mu_i qji = \frac{\mu_i}{\mu_j}\mu_j q_{ji} = \mu_j q_{ji}\frac{\mu_i}{\mu_j} = \mu_j p_{ji}$$

Hence $\mu_i p_{ij} = \mu_j p_{ji}$ for all $i \neq j$

Therefore, P is reversible with respect to μ
Then, μ is invariant probability distribution for P

3 Prove that P is irreducible and if μ isn't perfectly uniform then P is aperiodic

a) Prove P is irreducible
Since Q is irreducible then from any state i, we can reach state any state j in some t steps.

i) If $q_{ij} > 0$ for some i, j, then

$$p_{ij} = q_{ij}min(1, \mu_j/\mu_i) > 0 \text{ if } i \neq j$$
$$p_{ij} = 1 - \sum_{k \neq i} q_{ik}min(1, \mu_k/\mu_i) \geq 1 - \sum_{k \neq i} q_{ik} = q_{ij} > 0 \text{ if } i = j$$
$$\implies p_{ij} > 0$$

Hence, in chain P, from state i, we can reach state j in 1 step.

ii) If $q_{ij} = 0$ for some i, j, then under Q, from state i, we can go to state j in t steps. Let those steps be

$$i \to k_1 \to k_2 \to \dots k_{t-1} \to j$$

where k are different states from i and j

Therefore $q_{ik_1}, q_{k_1k_2}, \dots, q_{k_{t-1}j} > 0$
Since p_{ij} is a product of q_{ij} with a positive number, then $p_{ik_1}, p_{k_1k_2}, \dots, p_{k_{t-1}j} > 0$

Hence $i \to k_1 \to k_2 \to \dots k_{t-1} \to j$ under P.

Therefore, $i \to j$ under P in t steps

From i) and ii), then P is irreducible.

b) Prove that if μ isn't perfectly uniform then P is aperiodic
Since P is irreducible, then all states are of 1 class. Then it is enough to show that period(i) = 1 for only a state i.

If μ is not perfectly uniform, then assume μ_i is the largest entry in μ ($\mu_k < \mu_i, \forall k \neq i$). We have

$$p_{ii} = 1 - \sum_{k \neq i} q_{ik} min(1, \mu_k/\mu_i) > 1 - \sum_{k \neq i} q_{ik} = q_{ii} \geq 0$$
$$\implies p_{ii} > 0$$

Hence, from state i, if we can return to state i in n steps then we can also go back to state i in n+1 steps as the chain stay put at i in 1 step.

Since gcd(n, n+1) =1 then period(i) = 1

Hence, the chain is aperiodic.

4 Travelling salesman problem

Suppose we have N cities located at the vertices of a graph. The graph has n edge between cities if it is possible to travel between them, and we label with the travel time. A Hamiltonian circuit is a sequence of cities that starts and ends at some city, visits that city only at the beginning and end, and visits every other city exactly once. Our problem is to find the average travel time of circuits.

a) Check that Q is symmetric
Each state is a valid circuit
A state is something like

$$a = (1, 3, 7, 11, 15, 20, 4, 12, 17, 8, 19, 18, 2, 16, 5, 9, 6, 10, 14, 13)$$

If we randomly pick the 4th and 13th entries to swap (ie 11 and 2) and still get a valid sequence, this means a transition to a new state

$$b = (1, 3, 7, 2, 15, 20, 4, 12, 17, 8, 19, 18, 11, 16, 5, 9, 6, 10, 14, 13)$$

So the transition matrix Q entries are things like q_{ab}. There are a huge number of them (let say M). When a and b are different, q_{ab} is the reciprocal of 20-choose-2 when a and b differ by such a swap, and 0 otherwise.
By the same argument, q_{ba} is the reciprocal of 20-choose-2 when b and a differ by such a swap, and 0 otherwise.
Since there are finitely M number of valid circuits, then $q_{ab} = q_{ba}$

b) Coding steps and results
First, we labeled 20 cities. Then we picked 5 pairs of forbidden cities.

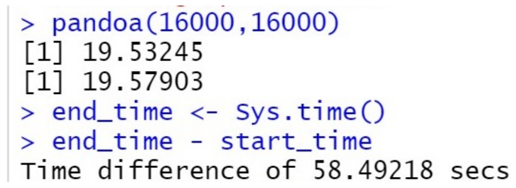
Secondly, we generated the 20 x 20 matrix, whose entries are random positive distances between 190 pairs of cities by exponential distribution with parameter $\lambda = 20$. Then we let the distances between each city to itself and between forbidden pairs to be 0.

From 20 cities, we have $\binom{20}{2} = 190$ pairs of cities, including 5 forbidden pairs. Then the number of valid pairs should be $190 - 5 = 185$ pairs

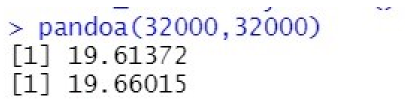
We created the initial sequence by randomly placing cities. Then we checked if this sequence is valid by checking whether any cities in the sequence to the next one in the sequence occur a distance of 0. If the sequence is invalid, then the function generate a new sequence. If it's valid, then we started swapping and checking for each of them.

Now, we write a loop which run the above function n times and another m times to find the mean travelling time for n, m circuits.

For $n = m = 16000$ the expected travelling time differ slightly, around 0.04. Then when we try the simulation for $n = m = 32000$, the expected values differ around 0.05. The computation took 58.49 seconds.
We can conclude n and m at 16000 is large enough.



```
> pandoa(16000,16000)
[1] 19.53245
[1] 19.57903
> end_time <- Sys.time()
> end_time - start_time
Time difference of 58.49218 secs
```



```
> pandoa(32000,32000)
[1] 19.61372
[1] 19.66015
```

APPENDIX

```
install.packages("expm")

library("expm")


#generate distance matrix

library(Matrix)


x<-Matrix(rexp(400),20)

M <- forceSymmetric(x)

for (i in 1:20){

  M[i,i]=0

}


#delete 5 pairs of cities

a<-c(3,5,7,8,1)

b<-c(4,11,18,13,10)

b1<-0

f<-0

for (i in 1:5){

  M[a[i],b[i]]=0

  M[b[i],a[i]]=0

}


#**********************************************************


#generate new series moves.

h<-c(1:20)
```

```
c<-sample(h)

c[21]<-c[1]

d<-c(2:20)

#********************************

#validation function

main<-function(k,a,b){

 b1=0

 e<-sample(d,2)

 u<-replace(k,c(e[1],e[2]),k[c(e[2],e[1])])


 #creates a valid sequence of cities

 while(b1!=0){

  b1=0  #reset the game

  for (i in 1:20) {

   g=u[i]

   j=u[i+1]

   if (M[g,j]==0){

     b1=1  #invalid cities

   }

  }

  #if u is not valid, we replace this iteration with a new city set

  if(b1==1){


   e<-sample(d,2)

   u<-replace(c,c(e[1],e[2]),c[c(e[2],e[1])])

  }

 }
```

```
  return(u)
}

#***************************************************
#sum distance of function
sumdistance<-function(c){
  for (i in 1:20){
    i=c[i]
    j=c[i+1]
    f=f+M[i,j]
  }
  return(f)
}
#***********************************************
#generate pandoa function
pandoa<-function(Nsim,Msim){
  g=0
  g1=0
  #run first n steps
  for (n in 1:Nsim){
    c<-main(c,a,b)#input a new valid series
    #sum a valid series cities
    f<-sumdistance(c)
    g[n]=f
    f=0
  }
  print (mean(g))
  #let the series run another m steps
  for (m in 1:Msim){

    #random pick two number then swap them to create new chain. new chain, the variable is u
    c<-main(c,a,b)
    f<-sumdistance(c)
    g1[m]=f
    f=0
  }
  #output the result from m steps
  print(mean(g1))
  #output of the simulation

}
start_time <- Sys.time()
pandoa(16000, 16000)
end_time <- Sys.time()
end_time-start_time

start_time <- Sys.time()
pandoa(32000, 32000)
end_time <- Sys.time()
end_time-start_time
```