From Definition to Application

An Simple Introduction to Markov Chain Monte Carlo

Yuyin Li, Qingli Zeng
Jiayu He, Yaokun Li

Dec. 21, 2018

Model Selection in Stochastic Modeling

1. Complex enough to capture the complexity of the phenomena in question, unlike I.I.D sequence of random variables, which is too restrictive.
2. Simple and structured enough to allow us to compute, unlike completely dependent among random variables, which is so hard to analysis.

Markov Process

There are 3 basic components of Markov Process.

1. A sequence of random variables $\{X_t, t \in \mathcal{T}\}$, where $\mathcal{T}$ is an index set, usually it is "**time**"
2. All possible sample values of $\{X_t, t \in \mathcal{T}\}$ are called "**states**", which are elements of a state space $\mathcal{S}$
3. "**Markov Property**": given the present value(information) of the process, the future evolution of the process is independent of the past evolution of the process.

Markov Chain

1. Introduced by Andrey Markov in paper published in 1906.
2. Derived by Markov Process, it is a happy medium between complete independence and complete dependence.
3. it allows for correlations among random variables and also have enough structure and simplicity for computations.

Markov Chain

Definition

A sequence of random variable $X_0, X_1, X_2, \ldots$ taking values in the **state space** $\{1, 2, ..., M\}$ is called a **Markov Chain** if for all $n \geq 0$,

$$P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = j | X_n = i)$$

Transition Matrix

To denote and simulate a Markov Chain, we need **Transition Matrix**, which is a **Stochastic Matrix**. It has many good properties; we will cover them in detail in our report.

Definition

Let $X_0, X_1, X_2, \ldots$ be a Markov chain with state space $\{1, 2, \ldots, M\}$, and let $q_{ij} = P(X_{n+1} = j | X_n = i)$ be the transition probability from state i to state j. The $M \times M$ matrix $Q = (q_{ij})$ is called the transition matrix of the chain.

Algorithm for Simulating a Markov Chain

Input

- initial distribution α.
- transition matrix $\mathbf{P}$.
- number of steps n.

Output

- $X_0, X_1, \ldots, X_n$.

Algorithm for Simulating a Markov Chain

Pseudo-code

```
1: Generate $X_0$ according to $\alpha$
2: **for** $i = 1$ to $n$ **do**
3:     $X_{i-1} = j$
4:     set $\mathbf{p} = \mathbf{P}[\,j\,][\ldots]$
5:     Generate $X_i$ according to $\mathbf{p}$
6: **end for**
```

Limiting and Stationary Distribution

Two perspectives of Limiting Distribution

- Long-term Probability that a Markov Chain hits each state
- Long-term Proportion of time that a Markov Chain visits each state

Definition of Stationary Distribution

Let $X_0, X_1, \ldots$ be a Markov chain with transition matrix $\mathbf{P}$. A **Stationary Distribution** is a probability distribution *pi*, which satisfies

$$\pi = \pi \mathbf{P}$$

Irreducibility and Aperiodicity

Definition of irreducibility

A Markov chain is called irreducible if all states communicate with each other.

Definition of Aperiodicity

For a Markov chain with transition matrix **P**, the period of state i, denoted $d(i)$, is the greatest common divisor of the set of possible return times to i. That is,

$$d(i) = \gcd\{n > 0 : \mathbf{P}_{ii}^{n} > 0\}.$$

if $d(i) = 1$, state i is said to be **aperiodic**.

Irreducibility and Aperiodicity

Irreducibility and **Aperiodicity** is very important for they are highly related to Markov Property, we will cover how to prove and build a Markov Chain with irreducibility and aperiodicity in our report.

Markov Chain Monte Carlo

Markov Chain Monte Carlo(MCMC) is a remarkable methodology, which utilize Markov sequence to effectively simulate from what would otherwise be intractable distributions.

Two most widely used algorithms: **Metropolis-Hastings** and **Gibbs Sampling**

Combinatorial Optimization

Definition

Combinatorial Optimization is a topic that consists of finding an optimal object from a finite set of objects

Importance

It is important in Operations Research, applied mathematics and theoretical computer science.

Combinatorial Optimization

Difficulty

In many such problems, exhaustive search is not tractable.

Examples

Graph Coloring(NP-Complete), Traveling Salespeople Problem(NP-Complete).

Metropolis-Hastings Algorithm

Basic Ideas

- Proposed by Nicholas Metropolis in 1953,
  further developed by Wilfred Keith Hastings in 1970.
- Start with any **irreducible** Markov chain on the state space.
  Then Modify it into a new Markov Chain with desired stationary distribution.
- Modification: moves are proposed according to the original chain, but the proposal may or may not accepted.
- Art: Choice of the probability of accepting the proposal.

Metropolis-Hastings Algorithm

```
Require:
    Stationary distribution $\pi = (\pi_1, \ldots, \pi_M)$;
    Original transition matrix $\mathbf{P} = (\mathbf{p}_{ij})$
    State $X_0$
Ensure:
    Modified transition matrix $\mathbf{P}' = (\mathbf{p}'_{ij})$
1: while NOT Convergence do
2:     if $X_n = i$ then
3:         propose a new state $j$ using the transition probabilities in the $i$th row of the original
           transition matrix $\mathbf{P}$
4:     end if
5:     Compute the acceptance probability $a_{ij} = \min\left(\frac{\pi_j \mathbf{p}_{ji}}{\pi_i \mathbf{p}_{ij}}, 1\right)$
6:     Filp a coin with $P = a_{ij}$, to decide accept the proposal or not.
7: end while
```

Metropolis-Hastings Algorithm

Example: Knapsack Problem

0-1 Knapsack problem is a famous combinatorial optimization problem, it can be solved by dynamic programming, with complexity $O(nW)$,The $O(nW)$ complexity does not contradict the fact that the knapsack problem is NP-complete, since W, unlike n, is not polynomial in the length of the input to the problem. The length of the W input to the problem is proportional to the number of bits in W, $\log W$, not to W itself.

Metropolis-Hastings Algorithm

Solving Knapsack Problem via MCMC

Consider the following Markov chain. Start at $(0, 0, \ldots, 0)$. One move of the chain is as follows. Suppose the current state is $x = (x_1, \ldots, x_m)$. Choose a uniformly random J in $\{1, 2, ..., m\}$, and obtain y from x by replacing x_J with $1 - x_J$. If y is not in C, stay at x. If y is in C, flip a coin that lands Heads with probability $\min\left(1, e^{\beta(V(y) - V(x))}\right)$. If the coin lands Heads, go to y; otherwise, stay at x.

Metropolis-Hastings Algorithm

Solving Knapsack Problem via MCMC

This chain will converge to the desired stationary distribution. But how should β be chosen? If β is very large, then the best solutions are given very high probability, but the chain may be very slow to converge to the stationary distribution since it can easily get stuck in **local modes**: the chain may find itself in a state which, while not globally optimal, is still better than the other states that can be reached in one step, and then the probability of rejecting proposals to go elsewhere may be very high. On the other hand, if β is close to 0, then it's easy for the chain to explore the space, but there isn't as much incentive for the chain to uncover good solutions.

Metropolis-Hastings Algorithm

Simulated Annealing

An optimization technique called **simulated annealing** avoids having to choose one value of β. Instead, one specifies a sequence of β values, such that β gradually increases over time. At first, β is small and the space C can be explored broadly. As β gets larger and larger, the stationary distribution becomes more and more concentrated on the best solution or solutions.
The name **"simulated annealing"** comes from an analogy with the annealing of metals, a process in which a metal is heated to a high temperature and then gradually cooled until it reaches a very strong, stable state; β corresponds to the reciprocal of temperature.

Metropolis-Hastings Algorithm

Solving Knapsack Problem via MCMC

Simulated Annealing is a way to balance **exploitation** and **exploration**. So at the beginning, we may want to set β a small value to explore, and when our knapsack is about to be full, we want to have a small β to exploit the space of knapsack.
Therefore, β should be positive correlated the how full our knapsack is.

Gibbs Sampling

Definition

Gibbs sampling is an MCMC algorithm for obtaining approximate draws from a joint distribution, based on sampling from conditional distributions one at a time: at each stage, one variable is updated (keeping all the other variables fixed) by drawing from the conditional distribution of that variable given all the other variables. This approach is especially useful in problems where these conditional distributions are pleasant to work with.

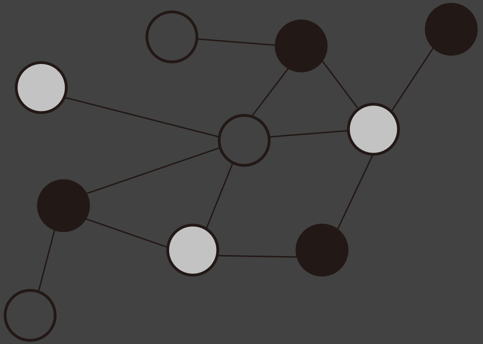
Gibbs Sampling

Two Major Kinds of Gibbs Sampler

Two major kinds of Gibbs sampler: **systematic scan**, in which the updates sweep through the components in a deterministic order, and **random scan**, in which a randomly chosen component is updated at each stage.

Gibbs Sampling

Solving Graph Coloring via MCMC



Gibbs Sampling

Solving Graph Coloring via MCMC

We would like to sample a random k-coloring of the entire graph; that is, we want to draw from the joint distribution of all the nodes. Since this is difficult, we instead condition on all but one node. If the joint distribution is to be uniform over all legal graphs, then the conditional distribution of one node given all the others is uniform over its legal colors. Thus, at each stage of the algorithm, we are drawing from the conditional distribution of one node given all the others: we're running a random scan Gibbs sampler.

MCMC is Widely Used

Solving Matrix Computation via MCMC

A paper in 14th International Conference on Computational Science. *Enhancing Monte Calro Preconditioning Methods for Matrix Computations*

MCMC is Widely Used

Solving Matrix Computation via MCMC

An enhanced version of a stochastic **SP**arse **A**pproximate **I**nverse **(SPAI)** pre-conditioner for general matrices is presented. This method is used in contrast to the standard deterministic pre-conditioners computed by the deterministic **SPAI**, and its further optimized parallel variant- **M**odified **SP**arse **A**pproximate **I**nverse **P**reconditioner (**MSPAI**).

Solving Matrix Computation via MCMC

Traditional Methods are Not Efficient Enough

Iterative solvers are used widely to compute the solutions of these systems and such approaches are often the method of choice due to their predictability and reliability when considering accuracy and speed. They are, however, prohibitive for large-scale problems as they can be very time consuming to compute. These methods are dependent on the size of the matrix and so the computational effort grows with the problem size. The complexity of these methods is $O(kn^2)$ for dense matrices in the iterative case and $O(n^3)$ for direct methods with dense matrices while solving SLAE if common elimination or annihilation schemes (e.g. Gaussian elimination, Gauss-Jordan methods) are employed.

Solving Matrix Computation via MCMC

Advantages of MCMC method

Monte Carlo (MC) methods can quickly yield a rough estimate of the solution. This is done by performing random sampling of a certain variable whose mathematical expectation is the desired solution. For some problems an estimate is sufficient or even favorable, due to the accuracy of the underlying data.
For example we would not need to process data with a higher precision than the one of the input data. In addition, Monte Carlo methods help to qualify the uncertainties while performing matrix inversion. For example, Monte Carlo approaches enable us to estimate the non-zero elements of the inverse matrix with a given precision, and with certain probability, and also enables us to estimate the structure of the inverse matrix.

Solving Matrix Computation via MCMC

How the MCMC Method Works

Therefore, we concentrate on Monte Carlo methods for matrix inversion (MI) that only require $O(NL)$ steps to find a single element or a row of the inverse matrix. Here N is the number of Markov chains and L is an estimate of the chain length in the stochastic process. These computations are independent of the matrix size n and also inherently parallel. Note that in order to find the inverse matrix or the full solution vector in the serial case, $O(nNL)$ steps are required.

Solving Matrix Computation via MCMC

How well the MCMC Method Works

The Algorithm is rather complicated and math-related, we will cover it in detail in our final report, but it works so well that everyone will be totally shocked.
We compare the 2 dimensions: Run Times and Residual.
We use **psmigr_3** and **Appu**, Psmigr 3 is a 3140×3140 matrix with 543162 non-zero entries, depicting data from US inter-county migration. Another set we use is Appu is a $14,000 \times 14,000$ matrix with $1,853,104$ nonzeros from NASAs app benchmark set.

Solving Matrix Computation via MCMC

How well the MCMC Method Works

For Run times:

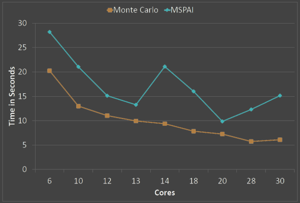


Figure: Run Time for psmigr_3

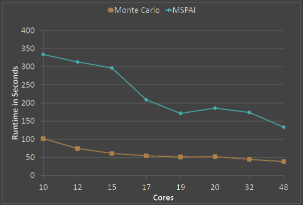


Figure: Run Time for Appu

Solving Matrix Computation via MCMC

How well the MCMC Method Works

For Residual:

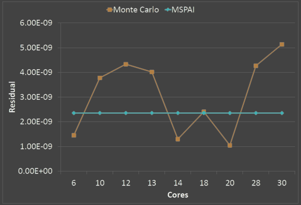


Figure: Residual for psmigr_3

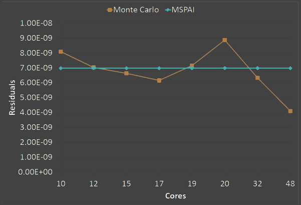


Figure: Residual for Appu

MCMC is Widely Used

Finding Minimum Node Separators via MCMC

A paper in Reliability Engineering and System Safety. *Finding Minimum Node Separators: A Markov Chain Monte Carlo Method*

MCMC is Widely Used

Finding Minimum Node Separators via MCMC

In networked systems such as communication networks or power grids, graph separation from node failures can damage the overall operation severely. One of the most important goals of network attackers is thus to separate nodes so that the sizes of connected components become small.
In this work, we consider the problem of finding a minimum α-separator, that partitions the graph into connected components of sizes at most αn, where n is the number of nodes.

MCMC is Widely Used

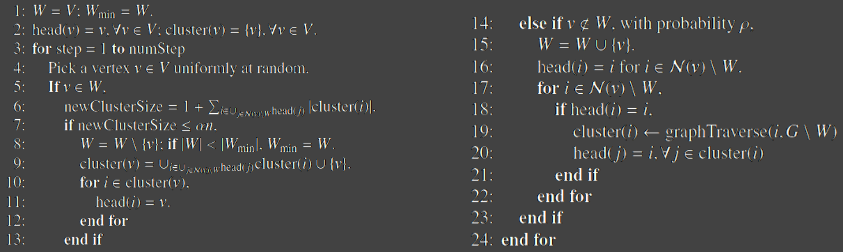
Finding Minimum Node Separators via MCMC

To solve the $\alpha-$*separator* problem, we develop a random walk algorithm based on Metropolis chain. We characterize the conditions for the first passage time (to find an optimal solution) of our algorithm.
We also find an optimal cooling schedule, under which the random walk converges to an optimal solution almost surely. Furthermore, we generalize our algorithm to non-uniform node weights. We show through extensive simulations that the first passage time is less than $O(n^3)$, thereby validating our analysis. The solution found by our algorithm allows us to identify the weakest points in the network that need to be strengthened. Simulations in real topologies show that attacking a dense area is often not an efficient solution for partitioning a network into small components.

Finding Minimum Node Separators via MCMC

Algorithm

Compared with Matrix Computation Algorithm we just mentioned, this algorithm is more easy to understand.



```
$W = V$; $W_{\min} = W$.
head($v$) = $v$, $\forall v \in V$; cluster($v$) = $\{v\}$, $\forall v \in V$.
for step = 1 to numStep
    Pick a vertex $v \in V$ uniformly at random.
    If $v \in W$,
        newClusterSize = $1 + \sum_{i \in \cup_{j \in \mathcal{N}(v) \setminus W} \text{head}(j)} |\text{cluster}(i)|$.
        if newClusterSize $\leq \alpha n$,
            $W = W \setminus \{v\}$; if $|W| < |W_{\min}|$, $W_{\min} = W$.
            cluster($v$) = $\cup_{i \in \cup_{j \in \mathcal{N}(v) \setminus W} \text{head}(j)} \text{cluster}(i) \cup \{v\}$.
            for $i \in$ cluster($v$),
                head($i$) = $v$.
            end for
        end if
    else if $v \notin W$, with probability $\rho$,
        $W = W \cup \{v\}$.
        head($i$) = $i$ for $i \in \mathcal{N}(v) \setminus W$.
        for $i \in \mathcal{N}(v) \setminus W$,
            if head($i$) = $i$,
                cluster($i$) $\leftarrow$ graphTraverse($i, G \setminus W$)
                head($j$) = $i, \forall j \in$ cluster($i$)
            end if
        end for
    end if
end for
```

Finding Minimum Node Separators via MCMC

Proof For the Algorithm

Proof for the algorithm is too long, so we leave it in our report.

Finding Minimum Node Separators via MCMC

How well the MCMC Method Works

We compare this algorithm with 2 baseline algorithm: **Highest-Degree-First** and **Greedy**.
These two algorithms all need to traverse the whole graph, so complexity is $O(n^3)$ And we use two graphs in real life as test set: **US Fiber Networks** and **Italian Power Grid**

Finding Minimum Node Separators via MCMC

Greedy

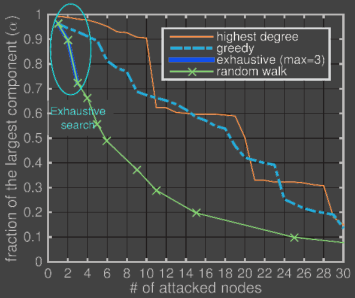
In each iteration, for each candidate vertex that is not in W, it computes the size of the largest connected component after adding the vertex. Then, the vertex with the largest marginal decrease is chosen. Break the tie randomly.

Highest-Degree-First

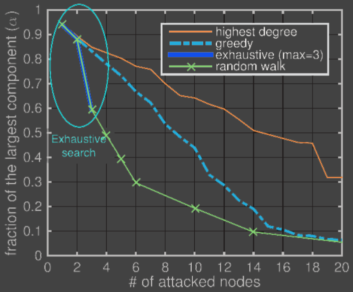
Attack vertices are chosen in the order of their node degree, i.e., the number of neighboring vertices. Break the tie randomly.

Finding Minimum Node Separators via MCMC

How well the MCMC Method Works



(a) In US fiber networks.



(b) In Italian power grid.

Thank you!