Lab L6: Dynamic processes on graphs - part II - Evaluated

Shaoyong Guo
Politecnico di Torino
s296966
s296966@studenti.polito.it

I. Introduction

Dynamic process theory is crucial in the study of many areas of network structure, so it has been applied to some interesting areas such as chemical reaction network theory, dynamic network analysis, etc. In this lab, we just study 3 types of model on finite graph, G(n,p) and the voter model over finite portion of Z^2 and Z^3. The challenge of this experiment is to choose the right data structure and how to optimise the algorithm. The experiment simulates a voter model with probabilistic binary state and evaluates the probability and time to reach a +1 consensus or a -1 consensus.

II. The simulation model

A. G(n,p) vector model.

Theoretically, in a graph G(n,P) model with n nodes, each vertex generates an edge with a certain probability, which is connected to other vertices in the graph. At the beginning of simulation, it is necessary to set the initial condition.

Stochastic Elements. The range of vectors is $[10^2,10^4]$.It will randomly select a number of vertices to generate a vector model, and then generate a variable number of edges between two vertices with some probability threshold 1/n. However, I also need a random Possion probability to assist the system in deciding whether it can generate an edge if the probability threshold is less than the random Possion probability.

Input Elements. The state vector of the vector model is a binary state [-1,+1] with an optional input probability distribution. During the simulation, each vertex will have a state that is consistent with the input probability distribution and can change over the time of the experiment.

B. 2-dimensional vector model

Input Elements. There is only one input element in the 2D vector model, the input probability of each vertex state. In my simulation, it is fixed at 0.51.

C. 3-dimensional vector model

Input Elements. The 3D vector model has the same input elements as the 2D vector model. The difference between the two vector models is the different spatial dimensions.

D. Data Structure

Different data structures have different ways of storing data. Matrix and dictionary are more suitable for storing large amounts of data than lists or tuples. There are three data structures in the experiment, lil-matrix, dok-matrix and COO(It is not coo-matrix).

lil-matrix In the SciPy library, scipy.sparse.lil-matrix refers to a specific sparse matrix representation called the LIL (List of Lists) format, which could generate sparse matrix is a matrix with specific probability.

dok-matrix scipy.sparse.dok-matrix in SciPy is a sparse matrix in Dictionary of Keys and Values (DOK) format. Each key in the dictionary represents a tuple (i, j), where i is the row index and j is the column index, and the corresponding value is the matrix element at that position.

COO Until now, most of the matrix types provided in the SciPy library have been 2D-based, including coo-martrix. The new sparse.COO can generate matrices in Dictionary format, just like coo-martrix. new sparse.COO supports 3D matrices, while coo-martrix does not support 3D vector models.

III. Algorithm

A. G(n,p) Model

In each iteration, all ”neighbourhood” sets are computed once, and when the number of +1’s is high, all values of the corresponding neighbourhood set positions in the lexicon change to 1. **How to generate a matrix.** In the G(n,p) model, a sparse matrix is created using a lil-matrix, and a dictionary is created where the key values are the indices of all the elements of the lil-matrix, and the value values are the random binary states.

Find out all neighbors of each vertex. When the random Poisson probability is less than the input probability, each position in the matrix is judged to produce state 1. In the simulation of G(n,p), 1 means that all points at position ”1” are connected. In the lil-matrix, all positions of 1 are ”neighborhoods” cluster.

Wake-up for each vertex. In each iteration, all ”neighbourhood” cluster are counted, and when the number of +1’s is higher, all values at the corresponding neighbourhood set position in the thesaurus are changed to one.
Iteration. The system will keep iterating until all values in the dictionary are either -1 or +1.

B. 2-dimensional vector model

How to generate a matrix. In a 2-dimensional vector model, the dok-matrix will directly generate a dictionary, with the key value being the index in the matrix and the value being the binary state value.

Find out all neighbors of each vertex. All points are placed in a grid with n rows and n columns, so that the neighbours of each point are the four points above, below, left and right of the current point.

Wake-up for each vertex. In these four neighbourhoods, when the number of +1’s is more, the status of the current point changes to +1 and vice versa.

Iteration. The system will keep iterating until all values in the dictionary are either -1 or +1.

C. 3-dimensional vector model

How to generate a matrix. In my simulation, I didn’t use the COO matrix because of the need to download an update to the sparse package. I have created a 3D array along with a dictionary with key as the index and value as the state value.

Find out all neighbors of each vertex. All points are placed in a 3D grid of n rows and n columns and n layers, so that the neighbouring points of each point are six points before, after, above, below, left and right of the current point.

Wake-up for each vertex. In these six neighbourhoods, when the number of +1’s is high, the state of the current point changes to +1 and vice versa.

Iteration. The system will keep iterating until all values in the dictionary are -1 or +1. Alternatively, the system will automatically stop iterating when the number of iterations reaches 3000.

IV. Results and Conclusion

```
**************************************************************************
**************************************************************************
****************The Result of G(n,p) Model*****************************
Original One Count In G(n,p) Vector Model: 3018
Original Negative In G(n,p) Vector Model: 1338
Final Numberof Ones In G(n,p) Vector Model: 4356
Final Number of Minus Ones In G(n,p) Vector Model: 0
Final P_neg_one of  G(n,p) Vector Model : 0.0
Final P_one of  G(n,p) Vector Model: 1.0
Final Iteration Time  of G(n,p) Vector Model: 2928
**************************************************************************
**************************************************************************
****************The Result of 2D Model*********************************
Original One Count In 2D Vector Model: 976
Original Negative One Count In 2D Vector Model: 9024
Final Numberof Ones In 2D Vector Model: 0
Final Number of Minus Ones In 2D Vector Model: 10000
Final Percentage of Ones In 2D Vector Model: 0.0
Final Percentage of neg Ones In 2D Vector Model: 1.0
Final T In 2D Vector Model: 0
**************************************************************************
**************************************************************************
****************The Result of 3D Model*********************************
Final P_1 of  G(n,p) Vector Model : 0.51
Original One Count In 3D Vector Model: 530
Original Negative One Count In 3D Vector Model: 470
Final Numberof Ones In 3D Vector Model: 492
Final Number of Minus Ones  In 3D Vector Model: 508
Final Percentage of Ones In 3D Vector Model: 0.492
Final Percentage of Minus Ones In 3D Vector Model: 0.508
Total Time (T) In 3D Vector Model: 4000
**************************************************************************
**************************************************************************
```

Fig. 1. The Quantile-Quantile of Empirical and Theoretical Degree