AI607: GRAPH MINING AND SOCIAL NETWORK ANALYSIS (FALL 2024)

Homework 1: The Kronecker Graph Generation

Release: September 20, 2024,
Due: October 4, 2024, 11:59pm

1 Introduction

This assignment aims to help you understand the principles of the Kronecker model, a powerful method for generating graphs that mimic real-world network patterns. The Kronecker model creates large graphs by repeatedly applying the Kronecker product to a small initial matrix, capturing essential characteristics observed in complex networks. In this assignment, you will implement two types of Kronecker models: the deterministic Kronecker model and the stochastic Kronecker model. You will generate graphs using both models and compare their properties, such as degree distribution, density, and singular values, with those of real-world graphs to explore how well the Kronecker graphs replicate the behaviors of real networks.

2 Kronecker graph

2.1 Kronecker Product and Power

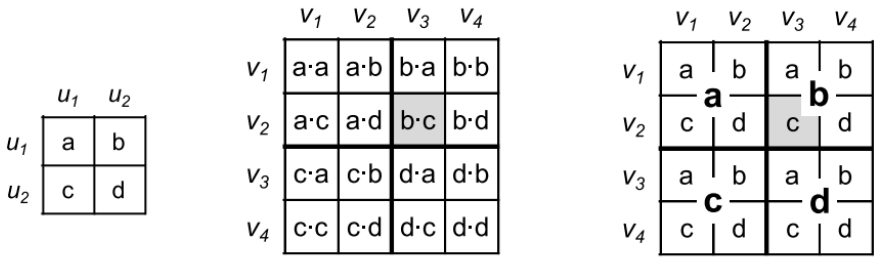
Given two matrices $\mathbf{A} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathbf{B} \in \mathbb{R}^{M_1 \times M_2}$, the Kronecker product $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{N_1 M_1 \times N_2 M_2}$ is constructed by multiplying each element of $\mathbf{A}$ by the entire matrix $\mathbf{B}$, resulting in the following form:

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} A_{11}\mathbf{B} & \cdots & A_{1N_2}\mathbf{B} \\ \vdots & \ddots & \vdots \\ A_{N_1 1}\mathbf{B} & \cdots & A_{N_1 N_2}\mathbf{B} \end{bmatrix} \tag{1}$$

The k-th Kronecker power of $\mathbf{A}$, denoted as $\mathbf{A}^{[k]}$, is defined recursively as $\mathbf{A}^{[k]} = \mathbf{A}^{[k-1]} \otimes \mathbf{A}$, with the base case $\mathbf{A}^{[1]} = \mathbf{A}$.

2.2 Deterministic Kronecker graph

A Kronecker graph of order k is generated using a small initial matrix, A_1. The adjacency matrix of the kronecker graph, denoted as $A_1^{[k]}$, is obtained by applying the Kronecker product to A_1 itself by k times.



(a) 2×2 Stochastic Kronecker initiator $\mathcal{P}_1$

(b) Probability matrix $\mathcal{P}_2 = \mathcal{P}_1 \otimes \mathcal{P}_1$

(c) Alternative view of $\mathcal{P}_2 = \mathcal{P}_1 \otimes \mathcal{P}_1$

Figure 1: Illustrations for the efficient sampling process of stochastic Kronecker product.

Toy Example: Let's start with a simple 2x2 initiator matrix:

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

- **Order 1 (** $k = 1$ **)**: The graph is just A_1.
- **Order 2 (** $k = 2$ **)**: Multiply A_1 with itself:

$$A_1^{[2]} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

This shows how the graph grows, repeating the pattern of A_1 at each step.

2.3 Stochastic Kronecker graph

Unlike the Deterministic Kronecker graph, the Stochastic Kronecker model allows the entries of the initiator matrix to take values between 0 and 1, representing probabilities instead of fixed connections. Each entry in the initiator matrix indicates the likelihood that a particular edge will appear in the graph. By applying the Kronecker power to this initiator matrix, we create a larger stochastic adjacency matrix, where each entry reflects the probability of an edge existing in the resulting graph.

Formally, let P_1 be an $N_1 \times N_1$ probability matrix, where each entry $P_1(i, j)$ represents the probability that edge (i, j) exists, with $P_1(i, j) \in [0, 1]$. The k-th Kronecker power, $P_1^{[k]}$, produces a larger probability matrix, where each entry $p_{uv} \in P_1^{[k]}$ specifies the probability of the edge (u, v). To construct a graph, we include the edge (u, v) with probability $P_1^{[k]}(u, v)$.

2.3.1 Efficient Generation of Stochastic Kronecker Graphs

Generating a stochastic Kronecker graph with N_1^k nodes using a naive approach requires O(N_1^{2k}) time, because it requires checking every possible edge between node pairs. To speed up this process when the target number of edges E is known, we can efficiently sample E edges by simulating the recursive structure

of the Kronecker product. The efficient method works by recursively selecting sub-regions of the matrix $P_1^{[k]}$ based on the probabilities given by the entries $P_1(i, j)$. This recursive selection continues for k steps until it reaches a specific cell in the large adjacency matrix $P_1^{[k]}$, where an edge is placed.

For example, in Figure 1, the edge (v_2, v_3) is placed by first selecting sub-region b with probability $\frac{b}{a+b+c+d}$, followed by selecting sub-region c with probability $\frac{c}{a+b+c+d}$. Typically, the total number of edges to be sampled matches the number of edges in the target graph, and if a sampled edge is sampled again, resampling is conducted.

3 Implementation

You will need to use Numpy [1] and Scipy [2] for this assignment. The Kronecker initiator matrices and their corresponding real-world networks are provided. For example, `data/input_cit-HepPh.txt` contains the initiator matrix for the stochastic Kronecker graph fitted to the HepPh citation network, as shown below:

```
0.990   0.440
0.347   0.538
```

Note that the size N of the initiator matrix can vary, allowing any sized matrix as an input. Your task is to implement code to compute $P_1^{[k]}$, which functions as the adjacency matrix for deterministic Kronecker graphs or as the probability matrix for edges in stochastic Kronecker graphs. From $P_1^{[k]}$, you will generate the corresponding graphs and analyze their network properties.

3.1 Kronecker graph

3.1.1 Deterministic Kronecker graph

First, load the initial matrix in the intializer of the class `Kprods`. Then, implement the `produceGraph` function in `Kronecker.py` to compute the k-th Kronecker power of the initiator matrix. You may use a recursive function to efficiently compute the kronecker product. For tracing the properties of the Kronecker graph, adjacency matrices P_i for all $i \in \{1, \cdots, k\}$ should be saved in the `self.adj_list` variable. The computed result (`self.adj_list`) should be returned by the `produceGraph` function. You should compare the properties of the resulting Kronecker graph with those of the 'cit-HepTh.txt' graph. We provide the Kronecker initiator in the file 'data/input_cit-HepTh_det.txt', and the exponent (k) should be 7.

3.1.2 Stochastic Kronecker graph

The provided code in `StochasticKronecker.py` defines the class `STKProds`, which efficiently generates stochastic Kronecker graphs using a recursive sampling method based on a given initiator matrix.

The class is initialized with three inputs: the number of Kronecker products k , a real-world adjacency matrix, and the path to the initiator matrix file. The `initializer` loads the initiator matrix, calculates its size, and estimates the target number of edges for each power level using the real-world adjacency matrix. It then sets up adjacency lists to store the generated Kronecker graphs.

[1] `https://numpy.org/install/`
[2] `https://scipy.org/install.html`

The `produceGraph` method generates the Kronecker graphs by recursively sampling edges until the target number of edges is reached for each power level. This is done using the `computeProb` method, which recursively selects sub-regions of the target Kronecker power matrix based on the probabilities defined by the initiator matrix entries. The process continues for k steps, narrowing down to a specific cell in the adjacency matrix, where an edge is placed. This efficient approach mirrors the recursive selection of sub-regions, as described in the Section 2.3.1, enabling the rapid sampling of edges without the need to check every possible node pair. You should compare the properties of the resulting Kronecker graphs with the 'cit-HepTh.txt' and 'cit-HepPh.txt' graphs. We provide the Kronecker initiators in the files 'data/input_cit-HepTh.txt' and 'data/input_cit-HepPh.txt', and the exponents (k) should be 14.

3.2 Analysis

In `Analysis.py`, we provide code to analyze three key properties of both Kronecker graphs and real-world graphs: degree distribution, density, and singular values.

3.2.1 Degree distribution

The degree of a node is defined as the number of edges connected to it. The `computeDegDist` function in `Analysis.py` calculates the degree distribution of the final graph. After the degree distribution is computed, the provided code will visualize the results. **You must include these visualizations in your report**.

3.2.2 Density

Density measures the ratio of edges to nodes in the graph, helping to observe how connectivity evolves across Kronecker power levels. The `computeDenDist` function in `Analysis.py` computes the density of the graphs at each Kronecker power level. The provided code compares how density changes as the Kronecker component increases in the case of Kronecker graphs, while real-world graphs show density growth over time as nodes are indexed chronologically. The code will generate visualizations of these density changes, and **you must include these plots in your report**.

3.2.3 Singular Values

Singular values are elements of the diagonal matrix produced by the Singular Value Decomposition (SVD) of a matrix. For more information, refer to this link. The `computeSvDist` function in `Analysis.py` calculates the top 100 singular values (or fewer, if fewer than 100 singular values are available) of the final graph. The function returns a dictionary where the keys are the ranks (i.e., the positions of the singular values in descending order, starting from 1), and the values are the corresponding singular values. The provided code will visualize these singular values, and **you must include these visualizations in your report**.

4 Test

You can test your implementation by executing the `main.py` with some options:

```
usage: main.py [-f FILENAME] [-k EXPONENT] [-o OUTPUTNAME]
               [-gf TARGETGRAPHFILENAME] [-s]

Optional arguments:
  -f FILENAME, --fileName FILENAME
        Specify the input file (e.g., input_cit-HepPh).
  -k EXPONENT, --k EXPONENT
        Set the exponent for the Kronecker product.
  -o OUTPUTNAME, --outputName OUTPUTNAME
        Specify the output file name for property analysis results.
  -gf FILENAME, --graphFileName FILENAME
        Specify the target real-world network file (e.g., cit-HepPh).
  -s, --stochastic
        Enable to use the Stochastic Kronecker Graph model.
```

For example, you can run `main.py` using the provided example as:

```
    python main.py -f input_cit-HepTh -k 14
                   -o citHepTh_stochastic -gf cit-HepTh -s
```

5 Notes

- Your implementation should run on TA's desktop within **2 minutes**.
- You may encounter some subtleties when it comes to implementation. Come up with your own design and/or contact Taehyung Kwon (taehyung.kwon@kaist.ac.kr) and Minyoung Choe (minyoung.choe@kaist.ac.kr) for discussion. Any ideas can be taken into consideration when grading if they are written in the *readme* file.
- Do not modify the code outside of the "TODO" regions. For example, do not import additional python libraries outside "TODO" regions.

6 How to submit your assignment

1. Create hw1-[your student id].tar.gz, which should contain the following files:
    - **main.py, Kronecker.py, StochasticKronecker.py, Analysis.py, utils.py**: Complete all the sections marked with "TODO" in these files.
    - **report.pdf**: Include your responses to Section 3.2 (Analysis), which involves analyzing and comparing the properties of the generated Kronecker graphs with those of real-world networks.
    - **readme.txt**: Contain the names of any individuals from whom you received help, and the nature of the help that you received. That includes help from friends, classmates, lab TAs, course staff members, etc. In this file, you are also welcome to write any comments that can help us grade your assignment better, your evaluation of this assignment, and your ideas.

2. All 7 files should be included in a **single folder**.
3. Make sure that no other files are included in the tar.gz file.
4. Submit the tar.gz file at KLMS (`http://klms.kaist.ac.kr`).