AI607: Graph Mining and Social Network Analysis (Fall 2021)

Homework 1. R-MAT: A Recursive Graph Generation

Release: September 10, 2021,
Due: September 27, 2021, 11:59pm

1 Introduction

How can we generate synthetic graphs that preserve structural properties observed in real-world graphs? In this assignment, you will understand and implement the *Recursive Matrix* (R-MAT) model, a simple but powerful graph generation model which captures the essence of real-world graph properties. You will find that with the appropriate choice of a few parameters, R-MAT well-describes the structural properties of real graphs.

2 R-MAT: A Recursive Model for Graph Mining

Here, we provide the design principles of the R-MAT model. For more details, see the original paper (`https://www.cs.cmu.edu/~christos/PUBLICATIONS/siam04.pdf`).

2.1 The Algorithm

Suppose we aim to generate a graph of N nodes whose adjacency matrix A is an $N \times N$ matrix with entry $A(i,j) = 1$ if the edge (i,j) exists, and $A(i,j) = 0$ otherwise. In the R-MAT model, it recursively subdivides the adjacency matrix into four equal-sized partitions and distribute edges within these partitions with an unequal probabilities. That is, starting from an empty matrix $\{0\}^{N \times N}$, it adds edges into the matrix one at a time, where Each edge is generated in one of the four partitions a, b, c, and d with probabilities p_a, p_b, p_c, and p_d, respectively, where $p_a + p_b + p_c + p_d = 1$ (see Figure 1). Once a partition is chosen, it again chooses one of the subdivided partition of the large one. This procedure is repeated until it reaches a simple cell (1×1 partition), where the edge is generated.

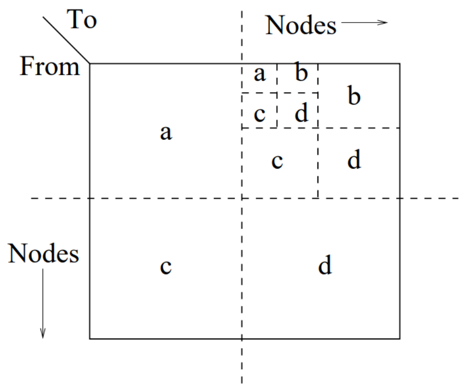

Figure 1: Four partitions a, b, c, and d of the adjacency matrix.

2.2 The Intuition Behind R-MAT

R-MAT intuitively and empirically meets several goals that a realistic graph generator should achieve. Typically, let $p_a \geq p_b$, $p_a \geq p_c$, and $p_a \geq p_d$.

- **R-MAT matches the degree distributions.** The skew in the distribution of edges between the partitions ($p_a \geq p_d$) leads to lognormals and power-law distributions.
- **R-MAT exhibits a community structure.** The partitions a and d represent separate groups of nodes which correspond to communities, and the partitions b and c represent the *cross-links* between the two groups. In addition, the recursive nature of the partitions naturally generates sub-communities within the existing communities.
- **R-MAT matches other real graph properties.** It empirically exhibits structural properties that are found in real graphs.

3 Implementation [30 points]

In this section, you will implement the R-MAT algorithm, which is one of the most popular graph generation methods. You are allowed to use Numpy [1] and Scipy [2] for the implementation. You are **NOT** allowed to use any other external libraries (Snap.py, NetworkX, etc.).

1. (30 points) Implement the `genRMat` function in the `main.py` which recursively adds edges in the adjacency matrix.

Given the number of nodes N (typically, we set N as the power of 2), the number of edges E, and four probability parameters p_a, p_b, p_c, and p_d, you should generate a *directed* graph of an adjacency matrix $A \in \{0,1\}^{N \times N}$. Note that there can be duplicate edges (i.e., edges which exist in the same cell in the adjacency matrix), but keep only one of them. You will add edges until the graph has E edges. You can test your implementation by executing `main.py` with following options:

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

```
usage: main.py [-h] [-n NUMNODES] [-e NUMEDGES]
               [-a PA] [-b PB] [-c PC] [-o OUTPUTNAME]

Generating a graph by R-MAT.

optional arguments:
  -n NUMNODES, --numNodes NUMNODES
        Select the number of nodes of the graph.
  -e NUMEDGES, --numEdges NUMEDGES
        Select the number of edges of the graph.
  -a PA, --pA PA
        The probability of the edge assignment in the partition a.
  -b PB, --pB PB
        The probability of the edge assignment in the partition b.
  -c PC, --pC PC
        The probability of the edge assignment in the partition c.
  -o OUTPUTNAME, --output OUTPUTNAME
        The file name of an output matrix.
```

Note that $p_d = 1 - p_a - p_b - p_c$. For example, you can run `main.py` as:

```
python main.py -n 256 -e 2000 -a 0.5 -b 0.2 -c 0.1 -o example
```

4 Analysis [70 points]

In this section, you will evaluate the graphs generated by the R-MAT method implemented in the previous section. You can evaluate them by executing `analysis.py` with following options:

```
usage: analysis.py [FILENAME]

Analyzing a graph generated by R-MAT.

positional arguments:
  INPUT          The file name of an input matrix
```

For example, you can run `analysis.py` as:

```
python analysis.py example.npz
```

We will analyze following three parameter settings:

(S1) $p_a = 0.25$, $p_b = 0.25$, $p_c = 0.25$ and $p_d = 0.25$

(S2) $p_a = 0.40$, $p_b = 0.20$, $p_c = 0.20$ and $p_d = 0.20$

(S3) $p_a = 0.70$, $p_b = 0.10$, $p_c = 0.10$ and $p_d = 0.10$

Note that setting **(S1)** subsumes the celebrated Erdős-Rényi model. For each setting $\mathbf{S}i$ ($i \in \{1, 2, 3\}$), you will generate two graphs `small` (N = 256 & E = 2,000) and `large` (N = 16,384 & E = 200,000) which we will denote by $G_{i,S}$ and $G_{i,L}$, respectively. Thus, you will generate & analyze six graphs in total ($G_{1,S}$, $G_{1,L}$, $G_{2,S}$, $G_{2,L}$, $G_{3,S}$, and $G_{3,L}$).

1. (30 points) Visualize the in-degree and out-degree distributions of the six graphs. The x-axis and y-axis should represent the node degrees and the number of nodes corresponding to the (in/out)-degree, respectively. You should implement `analyzeInDegrees` and `analyzeOutDegrees` functions.
2. (30 points) Visualize the singular value distribution of the six graphs. The x-axis and y-axis should represent the rank and the corresponding eigenvalues of the adjacency matrix, respectively. You can use the provided `getSingularValues` function.
3. (10 points) Provide the analysis in the report. How does the structure of the graph change depending on the parameters (p_a, p_b, p_c, and p_d)?

For all figures, both axes should be log-scaled. You can use the `drawLogLogPlot` function provided in `analysis.py`. Provide the figures in your report (**report.pdf**).

5 Notes

- Your implementation should run on TA's desktop within **10 minutes**.
- You may encounter some subtleties when it comes to implementation. Come up with your own design and/or contact Jihoon Ko (jihoonko@kaist.ac.kr) and Geon Lee (geonlee0325@kaist.ac.kr) for discussion. Any ideas can be taken into consideration when grading if they are written in the *readme* file.
- In `main.py` and `analysis.py`, do not modify the code outside of the "TODO" regions. For example, do not import additional python libraries outside "TODO" regions.
- You can freely use sparse matrix formats provided by SciPy. The detailed usage of the formats is in `https://docs.scipy.org/doc/scipy/reference/sparse.html`.

6 How to submit your assignment

1. Create hw1-[your student id].tar.gz, which should contain the following files:
   - **main.py** and **analysis.py**: these should contain your implementation.
   - **report.pdf**: this should contain your answers in Section 4 (Analysis).
   - **readme.txt**: this file should 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 **4 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`).