PROJECT: ANOMALY DETECTION IN TIME-EVOLVING GRAPHS

Overview: Anomaly detection refers to the problem of finding patterns in data which fail to conform to the expected standard. There are a variety of anomaly detection algorithms available, and the correct one to use depends on the type of anomaly you are trying to detect. A few examples are point anomalies, contextual anomalies, and collective anomalies. This project will be focused on anomaly detection in time evolving graphs. This is a challenging problem that has been researched in many different domains, including social networks, computer networks (e.g. TCP traffic), and road networks. Various methods have been explored, including time series models, similarity-based algorithms, and community-based algorithms. Evaluation of these approaches is often not straightforward because of the lack of a ground truth. However, we can compare the detected anomalies with normal time points in the dataset, or compare results among different methods in order to evaluate the performance of the methods indirectly.

Goal: Implement the given anomaly detection algorithm for real-word graphs. In order to do that, you will need to:

1. Implement the algorithm with an "obsession" for attaining the best performance possible.
2. Analyze, criticize, and evaluate the algorithm and turn in a report on your thoughts and findings.

Input: The sequential graph data sets and the paper describing the algorithm to be implemented.

Output: The implementation of the algorithm assigned to you and a report (details below).

Project Details:

- Read and understand the scientific publication assigned to you. If your publication has other emphases besides anomalous time point detection, then your report and implementation should focus only on the anomalous time point detection part.
- Implement the algorithm described in the publication using the programming language of your choice (C, C++, Java, Matlab, SAS, R, etc.).
- Implementation can be serial or parallel (whichever would be best suited for your particular algorithm). You can use Hadoop, openMP, MPI, etc.
- Measure the performance of the algorithm.

Project Plan:
You will have 1.5 weeks to complete the project. At the end of the 1.5 weeks you will submit both your source code and report, in compliance with the requirements outlined in the Submission Requirements.

Submission Requirements:

1. Algorithm code with detailed comments. If your program requires a special input format (different from the input format of the provided datasets), you should include a script to convert the provided format into the required format.

Your code should write the output to a file using the following format: the first line should indicate the total number of anomalies that your algorithm detected, and the following lines should be a list of anomalous time points. You will list all of the anomalous time points if there are fewer than 10, the top 10 if there are fewer than 100, or the top 10% if there are more than 100. By "top" anomalies we mean those that your algotithm considers to be the most anomalous. For example, if you are finding the similarity between pairs of graphs and those that fall below a threshold are considered anomalous, then you should output the time points that are farthest below the threshold. There should be one index (i.e. anomalous time point) per line.

2. README file with detailed instruction. It should contain at least the following:
   (a) Software that needs to be installed (if any) with URL's to download it from and instructions on how to install them.
   (b) Environment variable settings (if any) and OS it should/could run on.
   (c) Instructions on how to run the program (and the script to change the output format, if included).
   (d) Instructions on how to interpret the results.
   (e) Sample input and output files.
   (f) Citations to any software you may have used or any dataset you may have tested your code on.

3. **Project Report**
   Describe and discuss the algorithm from the scientific publication assigned to you. How could your algorithm be improved in order to be more robust and possibly identify more anomalies while not producing false positives? Could it be improved in terms of time or memory complexity? What kind of anomalies might your algorithm have trouble detecting? Why? Is your algorithm parameterized? Randomized? Does it start from a seed set of vertices? How does this affect the performance? Report how your algorithm performed on the provided data sets. Consider: what percentage of the total number of input graphs did your algorithm identify as anomalous? What does this say about the sensitivity of the algorithm? Are the detected time points very near each other or very spread out? Compare an anomalous timepoint to a normal time point. What differences do you see in the graph structure?

Grading Rubric:

Criteria	**Percentage**
Implementation	10%
Code executes and produces required results	40%
Project Report -	50%