M2 Data Science IP Paris, Data Stream Processing: Implementation of Sketch-Based Anomaly Detection in Streaming Graphs for River API

Victor Hoffmann, Julie Massé

January 7, 2024

1 Introduction

The goal of this project is to implement a streaming algorithm of a research paper following the guidelines from River. The researcher paper that we have implemented is [1]. This paper presents 4 algorithms that incorporates dense subgraph search to detect graph anomalies (anomalous edges or anomalous subgraphs) in constant memory and time and that outperforms state-of-the-art baselines on 4 real-world datasets.

2 Algorithms Review

The models in this paper are intended for application for intrusion detection, where abnormal behavior involves many connections between targeted machines. Dynamic graphs represent connections over time, with nodes like machines and edges like timestamped connections. The goal is to detect both anomalous individual connections (edge anomalies) and unusual communication patterns within dense subgraphs (graph anomalies). The proposed method uses higher-order sketching, extending the count-min sketch data structure, to enable efficient edge and subgraph anomaly detection in a continuous manner. The approach demonstrates higher accuracy and lower runtime compared to existing methods. The paper presents edge anomaly detection methods (**AnoEdge-G** and **AnoEdge-L**) and graph anomaly detection methods (**AnoGraph** and **AnoGraph-K**).

2.1 H-CMS: Higher-Order Count-Min Sketch

The algorithms use the concept of **Higher-Order Count-Min Sketch** (H-CMS), an extension of the popular Count-Min Sketch (CMS) used in streaming data structures. H-CMS uses multiple hash functions to map a multidimensional input to a tensor, thus providing more information than CMS. The three-dimensional H-CMS can hash two-dimensional features. The source node is hashed in the first dimension and the destination node in the other dimension of the sketch matrix, unlike CMS which will hash the entire edge into a one-dimensional line vector. This allows the transformation of dense subgraph detection into a dense submatrix detection problem, where the matrix size is a constant independent of the graph size. H-CMS provides similar estimation guarantees to CMS, even though it uses the same hash function for the source and destination nodes. However, the H-CMS overestimates the number of possible collisions but provides a reliable estimate with a high probability.

2.2 Edge Anomaly Detection

The algorithms **AnoEdge-G** (Global Edge Anomaly Detection) and **AnoEdge-L** (Local Edge Anomaly Detection) are used to detect edge anomalies by noticing if the received edge is part of a dense submatrix. AnoEdge-G finds a global dense submatrix and AnoEdge-L maintains and updates a local dense submatrix around the matrix element.

2.2.1 AnoEdge-G

For AnoEdge-G, the H-CMS counts are decayed periodically to simulate the forgetting of older information. Upon the arrival of an edge (u, v), the H-CMS counts are updated, and the density of a dense submatrix is calculating around the indices (h(u), h(v)). The higher density the higher likelihood of anomaly. To calculate the density, a submatrix is expanded by iteratively selecting rows and columns to maximize the sum and calculating the density at each step. Then the selecting rows and columns are removed. The maximum density over all iterations is the final anomaly score.

2.2.2 AnoEdge-L

AnoEdge-L uses a temporally decaying H-CMS for edge counts and maintains a mutable submatrix. The algorithm updates the submatrix greedily based on the incoming edges in order to maximize density. The score for an edge is computed with respect to the current submatrix. It has not been implemented in our project.

2.3 Graph Anomaly Detection

The algorithms **AnoGraph** and **AnoGraph-K** detect graph anomalies by mapping the graph to a higher-order sketch and identifying dense submatrices. AnoGraph employs a greedy approach, ensuring a 2-approximation guarantee on the density measure. AnoGraph-K strategically selects K matrix elements and greedily searches for a dense submatrix, exhibiting comparable performance to AnoEdge's density calculation procedure.

2.3.1 Anograph

AnoGraph uses the H-CMS data structure for storing edge counts, resetting when a new graph arrives. The algorithm updates the H-CMS counts for incoming edges and uses AnoGraph-Density algorithm to find a dense submatrix, where the anomaly score is the density of that submatrix. The procedure achieves a 2-approximation guarantee for the densest submatrix problem through a greedy approach to iteratively removing rows or columns with minimal sums.

2.3.2 Anograph-K

AnoGraph-K is very similar to AnoGraph but it uses the AnoGraph-K-Density procedure to calculate the density of the dense submatrix. It is based on the fact that matrix elements with a higher value are more likely to be part of a dense submatrix. Therefore, we take the K largest elements of the matrix and use the density calculation algorithm of AnoEdge-G. It has not been implemented in our project.

3 Implementation for River API

The `C++` codes of the algorithms presented in the paper were available in their GitHub Repository. In this project, we only implemented AnoGraph and AnoEdge-G, being the models with the best results respectively for Graph Anomaly Detection and for Edge Anomaly Detection. We adapted the code in `Python` following River guidelines.

As both models use the H-CMS data structure, it also had to be implemented. In the code, there are therefore three classes: `Hcms`, `Anograph` and `AnoEdgeGlobal`. Each class has its own functions detailed below.

3.1 `Hcms` Class

The H-CMS data structure had to be implemented since both Anograph and AnoEdge-G use Higher-Order Count-Min Sketch to detect anomalies. It has to be initialized with a two hyperparameters: the number of buckets and rows of the H-CMS Matrix. This class is not a machine learning model. Here are the following functions of the class:

- `hash`: it determines the bucket for an element in a specific row. From an element to be hashed an the index of the hash function, it calculates the hash value using hash coefficients and returns the hashed bucket index.

- `insert`: inserts an edge with a specified weight into the count matrix. From the two nodes from the edge and a weight, it iterates over the rows to insert the edge into the count matrix, determines the buckets using the hash function and increments the count for the edge in the count matrix.

- `decay`: Applies decay to the entire count matrix to decrease counts over time. It takes as input a decay factor and iterates over rows, buckets, and sub-buckets to decay each count value.

- `getAnoEdgeGlobalScore`: Computes the anomaly score for an edge base. It takes as input the source node and the destination node. After initialization, it iterates over the rows to calculate the density of each sub-matrix and update the minimum dense sub-graph score.

- `getAnographScore`: Computes the global anomaly score for the entire count matrix. It iterates over the rows to calculate the density of each sub-matrix and update the minimum dense sub-graph score.

3.2 `AnoEdgeGlobal` class

Each observation of a graph dataset represents a new edge in the graph arriving at a given time. AnoEdge-G is well-fitted for the River guidelines, since it can learn and evaluate each edge information individually. We were therefore able to create a `learn_one` and `score_one` function for this model. The other function in this class `getAnoEdgeGlobalDensity` is called when learning or scoring.

- `learn_one`: Update the `AnoEdgeGlobal` instance with a new edge. It extracts source, destination, and timestamp from the input dictionary. If the timestamp is greater than the last recorded time, it decays the count matrix using the specified decay factor. Then it inserts the new edge into the count matrix with a weight of 1, updates the last recorded time and returns the updated `AnoEdgeGlobal` instance.

- `score_one`: Calculate the anomaly score for a given edge. It extracts source, destination, and timestamp from the input dictionary. It calculates the anomaly score for the specified edge using the count matrix. And it returns the calculated anomaly score.

- `getAnoEdgeGlobalDensity`: Calculate the density-based anomaly score for a specific edge. After initialization flags for marked rows and columns, and slice sums for rows and columns, it marks the source and destination nodes and updates the slice sums accordingly. It finds the initial maximum row and column of the matrix input. Then it iterates through the matrix to find the maximum density. At each iteration, it updates the maximum row or column, marks it, and adjusts the matrix sum accordingly. Finally it calculates the density based on the marked rows and columns, and returns the final anomaly score for the specified edge.

3.3 `Anograph` class

AnoGraph was trickier to implement according to the River guidelines. Since AnoGraph detects graph anomalies and that each observation of the datasets represents an edge, observations needed to be accumulated up to a given time window (`time_window`) in order to build a graph. As a consequence, the labels of the datasets also had to be preprocessed: they were obtained by summing the anomalies over each time window. If the sum exceeds a given threshold (`edge_threshold`) then it is worth 1, and 0 otherwise.

Additionally, `Anograph` has no `score_one` function. This is due to several reasons. Firstly, asking the score given one observation (an edge) for a graph detection algorithm has no sense. Secondly the score cannot be given without necessarily learning the input. Indeed, AnoGraph accumulates data in H-CMS matrix over time. When the

time window is exceeded, complete graph is created and density score is calculated for an H-CMS sub-matrix. There is therefore only one `get_score` function which has no arguments and allows to get the score given the learned observations. Here are the functions of the `Anograph` class. It is important to note that the `pickMinRow, pickMinCol, getMatrixDensity` and `getAnographDensity` functions are called in the `learn_one` function.

- `learn_one`: Update the `Anograph` instance with a new edge. It extracts source, destination, and timestamp from the input dictionary. If the timestamp falls into a new time window, it updates H-CMS data structure with the current batch of edges. Density score is calculated for an H-CMS sub-matrix.

- `get_score`: Get the current anomaly score.

- `pickMinRow`: Pick the row with the minimum sum from the matrix. It iterates over rows to find the one with the minimum sum.

- `pickMinCol`: Pick the column with the minimum sum from the matrix. It iterates over columns to find the one with the minimum sum.

- `getMatrixDensity`: Compute the density of the submatrix specified by parameters. It iterates over rows and columns to calculate the density of the selected submatrix.

- `getAnographDensity`: Compute the Anograph density of the matrix. After initialization, it iterates to pick and exclude rows or columns to maximize density and updates output with the maximum density obtained.

3.4 Scores

The score given by Anograph and AnoEdge-G represent the density of the submatrix. The higher the density, the more anomalous is the observation. The score can be anything between 0 and ∞ and can't be normalized between 0 and 1. This is not a problem for scikit-learn's Area Under the ROC Curve function `roc_auc_score` which just needs the order of scores. But this doesn't work with River's AUC which needs a value between 0 and 1. We tried to use a MinMaxScaler in the pipeline but this gave completely wrong results. This is why we weren't able to use a streaming metric of River and had to use scikit-learn's `roc_auc_score` after the model had learned and scored on the whole dataset.

4 Experiments results

For similar parameters, we obtain the same accuracy results which shows that the algorithms have probably been implemented well. But we observe that the running times of our implementation are much higher than the values of the source code. This can be attributed to several reasons.

The code was initially in `C++` known to be one of the fastest languages in the world as opposed to Python. We tried to speed up the process with libraries such as `Numpy`, which is known to speed up matrix operations, but it surprisingly made the computation time longer. This can be due to the fact that we don't do any classical matrix operations such as matrix addition, multiplication or invertion which is where `Numpy` truly shines. A `Cython` implementation of the code might be a good idea to have more efficient computing.

An other reason why the compution time is significantly longer is the fact that we trained our model on a small laptop with very few computation power compared to the machine they used in the article to train their models.

Given the long execution time, we only tested performance on `DARPA` and `ISCX-IDS2012` datasets. Results for AnoGraph are shown in Table 1 and results for AnoEdge-G are shown in Table 2.

Dataset	AUC Code	AUC Paper	Running time Code	Running time Paper
DARPA	0.835	0.835	53.66s	0.3s
ISCX-IDS20124	0.969	0.950	55.54s	0.5s

Table 1: AnoGraph results comparison for DARPA and ISCX-IDS20124 datasets. The Area Under the ROC Curve of our code is very close to the Area Under the Curve of the paper. However, the running time is much longer which might be due to the performance drop of Python when compared to C++ in addition to our laptop's rather poor computational power.

Dataset	AUC Code	AUC Paper	Running time Code	Running time Paper
DARPA	0.969	0.970	5684.66s	28.7s
ISCX-IDS20124	0.954	0.954	1343.42s	7.8s

Table 2: AnoEdge-G results comparison for DARPA and ISCX-IDS20124 datasets. The Area Under the ROC Curve of our code is very close to the Area Under the Curve of the paper. However, the running time is much longer which might be due to the performance drop of Python when compared to C++ in addition to our laptop's rather poor computational power.

5 Conclusion

The performance is similar with the paper which is satisfactory for the moment. However the execution time could be improved. We could seek to optimize the code by implementing a Cython version. If we wanted to go further we could also implement the two other algorithms: AnoGraph-K and AnoEdge-L even though they led to slighlty poorer results in the article. AnoEdge-L would be particuarily relevant, since it has a decreased computation time compared to AnoEdge-G, due to its local nature. Furthermore we could test our models on all of the datasets in the paper if we have more computation power and memory.

References

[1] WADHWA Mohit KAWAGUCHI Kenji et al. BHATIA, Siddharth. Kenji, et al. sketch-based anomaly detection in streaming graphs. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 93–104, 2023.