# A simple graph embedding for anomaly detection in a stream of heterogeneous labeled graphs

Abd Errahmane Kiouche, Sofiane Lagraa, Karima Amrouche, Hamida Seba

# A Simple Graph Embedding for Anomaly Detection in a Stream of Heterogeneous Labeled Graphs

This is a preprint: the final paper is published in PRL
https://doi.org/10.1016/j.patcog.2020.107746

A. Kiouche[1], S. Lagraa[3], K. Amrouche[2], H. Seba[1]

**Abstract**

In this work, we propose a new approach to detect anomalous graphs in a stream of directed and labeled heterogeneous graphs. The stream consists of a sequence of edges derived from different graphs. Each of these evolving graphs represent the evolution of a specific activity in the monitored system whose events are acquired in real-time. Our approach is based on graph clustering and uses a simple graph embedding based on substructures and graph edit distance. Our graph representation is flexible and allows to update the graph vectors as soon as a new edge arrives. This allows the detection of anomalies in real-time which is an important requirement for sensitive applications such as cyber-security. Our implementation results prove the effectiveness of our approach in terms of accuracy of detection and time processing.

*Keywords:* Graph anomaly detection, Graph stream, Graph embedding, Graph edit distance

## 1. Introduction

Anomaly detection is a fundamental problem encountered in many real-world applications mainly related to surveillance and monitoring such as cyber-security, health, and finance. Generally, it consists of detecting data which are significantly different from benign or normal data. In this context, graphs are increasingly involved mainly because events in monitored systems are represented by graphs to capture dependencies among event flows and trace-back the root causes of anomalies in order to understand them [1]. In this case, the challenge of anomaly detection involves identifying those graphs which are different from the graphs representing normal events observed by the system. We recall that

[1]Université de Lyon, CNRS, Université Lyon 1
LIRIS, UMR5205, F-69622 Lyon, France.
[2]Laboratoire de la Communication dans les Systèmes Informatiques(LCSI)
École nationale Supérieure d'Informatique, Alger, Algérie.
[3]Université de Luxembourg
SnT - Interdisciplinary Centre for Security, Reliability and Trust.

a graph $G = (V, E)$ is a data modeling tool consisting of a set $V$ of vertices and a set $E$ of edges that connect vertices. Vertices represent objects and edges represent relationships between them. Detecting anomalies in graph data has been subject to several investigations [2, 3]. Proposed solutions are tightly related to the underlying applications, to the kind of graphs these applications deal with and how anomalies are defined within them. Broadly speaking, graph based anomaly detection approaches can be categorized into two main classes: those designed for static graphs and those designed for dynamic graphs. With anomaly detection in static graphs, the whole structure of the graph is known and the problem consists in spotting anomalous graph parts [4, 5], i.e., vertices [6, 7], subgraphs [8, 9], etc.

The aim behind the use of dynamic graphs is to have snapshots of evolving systems or objects which is a natural representation for several applications especially communication networks and social streams. Most works concern temporal anomalous pattern detection in time-evolving graphs, i.e., a sequence or stream of static graphs [10, 11]. The problem in this case consists to compare consecutive graphs in the stream [12, 13, 14]. However, streams of static graphs are not adapted for massive data and real-time applications [15]. In fact, the comparison of two consecutive graphs generally requires that the graphs are loaded in main memory. In addition, for several applications such as real-time surveillance, the complete structure of the graphs is unknown and acquired in real-time. Moreover, a surveillance application may involve the simultaneous monitoring of several and different activities of the system. This means that the anomaly detection algorithm must deal with several graphs that evolve independently, but simultaneously according to the activity they represent in the system. So, we deal with interlaced graph sequences as depicted in Figure 1. In

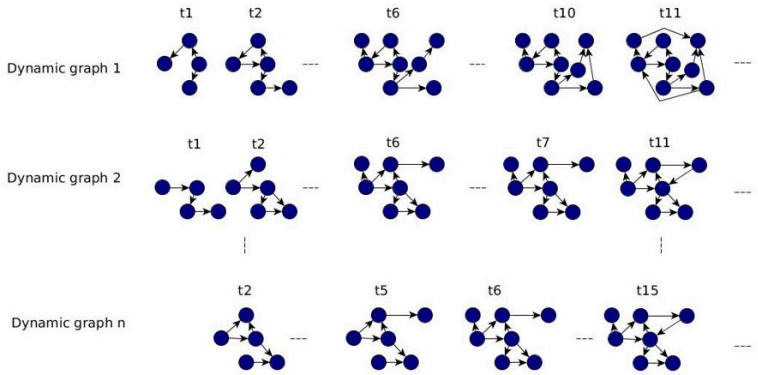



Figure 1: Dynamic graphs evolving through time.

this case, anomalies are detected on a stream of edges where the stream consists of individual graph edges instead of entire graph objects [15]. Consecutive edges in the stream may not belong to the same graph. This issue raises several challenges [16]:

- Graph heterogeneity: The graphs are heterogeneous because they represent different activities in the system. Anomaly detection must concern all the evolving graphs.
- Real-time anomaly detection: The system requires real-time anomaly detection. Each edge arriving in the stream modifies an activity graph. The resulting graph must be checked for anomaly.
- Memory efficiency: The stream of edges is continuous and represents voluminous data that cannot be stored in its entirety in main memory. This requires a reduced representation of the evolving graphs that can be incrementally maintained with the arriving edges.

In this paper, we deal with these issues using a simple vector representation of the graphs. The proposed representation is flexible and enables graphs to be incrementally maintained with the arrival of new edges, without having to save all the edges in the main memory. Moreover, anomalies are detected in real-time following the arrival of new edges. The main contributions of this work are:

- We propose a new approach for detecting abnormal directed and labeled heterogeneous graphs over streaming edges using graph edit distance.
- We propose a simple graph embedding approach for training and classifying abnormal graphs. The representation is flexible and allows to update the graphs efficiently as soon as a new edge arrives.
- We propose a real-time application for detecting abnormal events in cybersecurity data.

We evaluate and compare our approach with StreamSpot [16] algorithm. Our approach outperforms the existing one in terms of processing, memory consumption and detection.

The remainder of this paper is organized into five sections. Section 2 presents the issue of anomaly detection in a stream of edges in more details and review related work on this domain. Section 3 provides a description of the proposed approach and analyses its complexity. Section 4 presents the results obtained through experimentation. Section 5 concludes the paper.

## 2. Problem statement and related work

Let us consider a stream of directed and labeled edges belonging to different graphs. The stream is continuous and feeds a set of evolving graphs. The graphs evolve only by growing, with new edges connecting existing vertices or new ones. So, we deal with labeled and directed multigraphs. Two edges having the same endpoints, the same direction and the same label can be distinguished by their timestamps, i.e., the time at which they appear in the stream. Such multiple edges represent, for example, successive authentication attempts originating from the same host. Each of these graphs can be represented by 4-tuple

$G = (V, E, \Sigma_V, \Sigma_E)$ where $V$ is the set of vertices, $E$ is the set of edges, $\Sigma_V$ is the set of vertex labels, and $\Sigma_E$ is the set of edge labels. Each edge in the stream is represented by a 6-tuple:
<source vertex, $l_s$, destination vertex, $l_d$, $l_e$, $ID\ graph$ >
where $l_s$, $l_d$, and $l_e$ represent the labels of the source vertex, the destination vertex, and the edge respectively. Edges that share the same $ID\ graph$ belong to the same graph. Furthermore, the edges coming from different graphs can be interwoven and thus several graphs may evolve simultaneously. The aim of this work is to detect abnormal graphs at any moment $t$. This problem is defined as follows:

**Given:** a continuous stream of directed labeled edges belonging to different graphs.

**Find:** the evolving graphs that deviate from the expected patterns.

An expected pattern is a benign or normal graph known to the system and acquired during a training phase. An abnormal graph is defined as being a graph which is significantly different from such benign prototype. As such, detecting anomalies could be seen as a comparison/classification issue: as the graphs evolve with the arrival of new edges, they are classified as normal or abnormal according to their similarity to some graph prototypes which represent normal behavior in the system. The main underlying question is therefore: how to calculate similarity between graphs?

Comparing two graphs is a complex problem with exponential time solutions in the general case [17]. To obtain approaches for polynomial comparison, the most common methods can be categorized into two classes according to whether the comparison is undertaken in the graph space or in the vector space [18]. In the graph space, the main approach is to decompose the two graphs to be compared into simpler substructures and to compare the obtained substructures [19]. There are several methods for calculating similarity between graphs using their substructures, but the most common are based on Graph Edit Distance (GED). GED defines the similarity between graphs by the minimum costing sequence of edit operations that convert one graph into the other [20]. An edit operation is either an insertion, a suppression or a re-labeling of a vertex or an edge in the graph. A cost function associates a cost to each edit operation to measure the strength of each edit operation.

Among the decompositions used with GED, we can cite without being exhaustive: the decomposition into stars [21], trees [22], paths [23] or hybrid decompositions [24]. The fastest approximations of GED are of polynomial complexity [25]. However, GED must be recalculated upon the arrival of each new edge, which is costly to be considered in the context of a graph stream. It is interesting to see that in spite of the popularity of GED as a graph comparison tool in general, very few GED-based solutions are used for graph streams.

Comparing graphs in vector space aims to use the similarity measures and distances available in this space such as the Euclidean distance, the Jaccard distance, and cosine similarity. Several techniques and approaches have been proposed to transform graphs into vectors. This is called "Graph Embedding". Graph Embedding is a technique that transforms graphs into vectors in such

way that similar graphs are transformed into close vectors in the vector space [26]. These methods transform the whole graph into a vector (not to be confused with methods of graph embedding which transform the graph objects, i.e., vertices, substructures, ..., etc., into vectors). Graph embedding methods based on graph spectral theory are not suitable for our problem because they are designed for plain graphs and do not take into account edge and vertex labels. Sub-pattern based graph embedding techniques such as graph kernels [27, 28] cannot be directly applicable to stream of graphs as they require knowing all substructures of all the graphs to be classified. This cannot be done in a streaming scenario since graphs grow over time [16]. A Graph embedding using GED is also proposed [29]. It represents a graph $G$ by a vector $E_G = (d_1, ..., d_M)$ containing the GED between the graph $G$ and $M$ prototype graphs selected during the training phase. However, this method is not effective for a streaming scenario as the computational time required to update GED incrementally (i.e., upon the arrival of each new edge) is at least of quadratic complexity $\mathcal{O}(n^2)$ where $n$ represents the number of edges in the graph [30, 31]. Therefore, the required time to update the vector of graph $G$ at the arrival of a new edge is at least $\mathcal{O}(M * n^2)$.

Consequently, the main issue for anomaly detection in a graph stream is defining a graph similarity measure that can be computed incrementally while maintaining a low-memory costing representation of the graphs.

Classy [32], which addresses the problem of detecting malwares in a stream of call graphs, i.e., directed graphs representing calls between programs, uses an approximation of GED based on Simulated Annealing. However, it is designed for sequences of graphs and does not deal with incremental arrival of edges. Other methods extend concept used to detect anomalies in non-structured data streams such as sketching [33]. Spotlight [34] is a randomized sketching based-approach for detecting anomalies in a stream of time evolving directed labeled bipartite graphs. A graph sketch contains the total edge weights of $K$ specific directed subgraphs chosen independently and uniformly at random. The distance between graphs is defined as the squared Euclidean distance between their sketches. In Spotlight, anomalous graphs are defined as graphs concerned with a sudden disappearance or appearance of a dense subgraph. The strong point of this approach is that it represents each graph with a limited size sketch, where each dimension represents the sum of edges weights of one region (subgraph) in the graph. Anomalous graphs can be detected by spotting sketches which are far away from normal ones in sketch space. To allow the update of the sketchs incrementally at the arrival of the new edges, Spotlight uses hash functions which ensure that each vertex remains mapped to the same subgraph all over the time. Spotlight is designed for plain bipartite graphs and detects only one specific type of anomalies (sudden (dis)appearance of a dense subgraph). In [15], the authors introduce the first anomaly detection approach dealing with a stream of edges. They consider a stream of non labeled non directed edges and detect anomalous edges in the stream using a Count-Min sketch [35]. StreamSpot [16] is designed for online detection of abnormal graphs in a stream of directed and labeled edges derived from different graphs. Similarly to the construction of vector

representations of text documents by shingling them into $k$-grams, StreamSpot decomposes a graph into a set of $k$-shingles using tree-based $k$-grams [22]. A graph is represented by its vector of frequencies of $k$-shingles. As the number of $k$-shingles is infinite, StreamSpot relies on SimHash [36] which preserves the cosine similarity to map the high dimensional vectors of shingles into fixed dimensional vectors. StreamSpot is fast, memory efficient and succeeds in maintaining incrementally the graph representations. However, it has two main weaknesses: 1) to maintain the graph representation incrementally, it retains in memory a limited number of edges. When this memory is filled, Sreamspot deletes old edges to store the new incoming ones. As a result, a part of each graph is lost and graphs may not be classified correctly. 2) The cosine similarity used for graph comparison is not accurate. In fact, it depends only on the number of common substructures between the two graphs to be compared, and it makes no comparison between the sub-structures. Therefore, it looses accuracy when the sub-structures are too large. To solve this problem, StreamSpot divides shingles into smaller substructures called chunks. However, the choice of the size $C$ of a chunck impacts significantly the accuracy of the similarity measure. Indeed, a small $C$ makes most pairs of graphs appear similar, while a large $C$ makes them less similar. The drawback of this solution is therefore the re-calibration of $C$ upon the arrival of a new type of benign graph. SedanSpot [37] is a randomized approach for detecting anomalous edges in a stream of edges deriving from one graph. Anomalous edges are those which connect two sparsely connected parts of the graph. SedanSpot is a real time approach for detecting anomalous edges using sub-linear memory. However, it's designed to detect anomalous edges in a stream of edges of one graph and cannot be applied to a stream of heterogeneous graphs.

## 3. Label Oriented Graph Embedding for Anomaly Detection

In this section, we present LEADS, a Label oriented graph Embedding for Anomaly Detection in a Stream of directed labeled edges belonging to heterogeneous graphs. LEADS takes advantage of the labeling of edges and vertices and the direction of edges in the stream to construct a simple vector representation of the incoming heterogeneous graphs. LEADS is based on a directed label representation of a graph called a *label structure* and uses a GED based distance to compare these representations. The proposed graph embedding is based on these concepts.

### *3.1. Label Structures*

The proposed embedding relies on a label representation of the vertices of the graph called *Label Structures* and defined as follows:

**Definition 1 (Label Structure).** *Given a multigraph $G = (V, E, \Sigma_V, \Sigma_E)$, a label structure LS in G for a vertex v is a 3-tuple $(\ell_v, L^-, L^+)$ where $\ell_v$ is the label of v, $L^-$ (resp. $L^+$) is an vector of size $|\Sigma_E|$ where $L^-(i)$ (resp. $L^+(i)$) is the number of edges incoming to (resp. outgoing from) v and labeled with i.*

So, each graph can be represented by a set of LSs: an LS by vertex. Label structures are similar in spirit to stars [29]. Figure 2 illustrates this representation on an example.

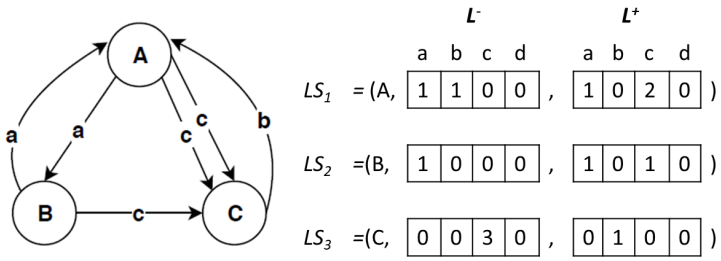



Figure 2: Example of label structures.

To compare two LSs, we rely on the number of edit operations required to transform one LS into the other. We consider three main kinds of edit operations having the same cost (equal to 1): $edit_\ell$ refers to the modification, addition or suppression of the vertex label of a label structure. $edit^-$ refers to the modification, addition or suppression of incoming labels in the label structure, and $edit^+$ refers to the modification, addition or suppression of outgoing labels in the label structure. The distance is defined as follows:

**Definition 2.** *Let $LS_1 = (\ell_1, L_1^-, L_1^+)$ and $LS_2 = (\ell_2, L_2^-, L_2^+)$ be two label structures, the distance between them is :*

$$d(LS_1, LS_2) = edit_\ell(LS_1, LS_2) + edit^-(LS_1, LS_2) + edit^+(LS_1, LS_2) \quad (1)$$

*with*

$$edit_\ell(LS_1, LS_2) = \begin{cases} 0 & if\ \ell_1 = \ell_2 \\ 1 & else \end{cases} \quad (2)$$

$$edit^-(LS_1, LS_2) = \max(\sum_{i=1}^{\Sigma_E} L_1^-(i), \sum_{i=1}^{\Sigma_E} L_2^-(i)) - \sum_{i=1}^{\Sigma_E} \min(L_1^-(i), L_2^-(i))$$

$$edit^+(LS_1, LS_2) = \max(\sum_{i=1}^{\Sigma_E} L_1^+(i), \sum_{i=1}^{\Sigma_E} L_2^+(i)) - \sum_{i=1}^{\Sigma_E} \min(L_1^+(i), L_2^+(i))$$

**Theorem 1.** *Under a unit cost, the distance given by Definition 2 is the exact graph edit distance between label structures.*

Proof. Let $s_1 = (\ell_1, L_1^-, L_1^+)$ and $s_2 = (\ell_2, L_2^-, L_2^+)$ be label structures. To prove that $d(s_1, s_2)$ is the exact graph edit distance between the two label structures, we have to prove that it gives the cost of an optimal sequence of edit operations that matches $s_1$ to $s_2$. By construction, a label structure contains

only one vertex. So, $edit_\ell(s_1, s_2)$ is the optimal cost for matching the vertices of the label structures. For the incoming edges (resp. the outgoing edges), $edit^-(s_1, s_2)$ (resp. $edit^+(s_1, s_2)$ computes the minimum number of edge relabelling/edge suppression/edge addition to have similar label structures which is the difference between the largest degree and the number of common edges between $s_1$ and $s_2$. This cost cannot be reduced. It is optimal.

### *3.2. Graph Embedding*

Our graph embedding method relies on label structures. We first choose a set of label structures that correspond to the benign activities of the system, i,e., to normal graphs. We use these label structures, that correspond to the expected behaviour of the system, as prototypes. This means that the label structures of the arriving graphs in the stream will be compared to these prototypes using distance $d$. Finally, we use the results of these comparisons as entries in a vector representation of the incoming graphs. In the following, we will first show how these prototypes are selected. Then, we describe how we use them to construct our graph embedding.

#### *3.2.1. Selecting the Prototype Label Structures*

The prototype label structures must correspond to the expected behaviour of the system. So, they must be taken from benign graphs that do not contain anomalies. The number $M$ of prototype label structures to be selected is determined by experiments (see Section 4). Assume we have $S$ different types of graphs (these types of graphs may represent for example the different activities that are monitored in the system or the various sources of the graphs). We use a set of benign graphs from each type (i.e., class) of graphs and we decompose them into label structures. Then, we use the Spanning Prototype Selection Class-wise strategy (SPS-C) [29] to select our prototypes. SPS-C selects $\frac{M}{S}$ prototype label structures from each class $C$. Let $P$ be the set of prototype label structures. With SPS-C, $P$ is computed iteratively by selecting a new prototype $p$ at each iteration. We denote by $P_i$ the content of $P$ at iteration $i$. $P_1$ contains the median label structure of the class. Each additional prototype $p$ selected by SPS-C is the furthest label structure from the already selected ones as follows:

$$P_i = \begin{cases} Median(C) & if\ i = 1 \\ P_{i-1} \cup \{p\} & if\ 1 < i < \frac{M}{K} \end{cases} \tag{3}$$

$$with\ p = \underset{LS \in C \setminus P_{i-1}}{\arg\max} \min_{p \in P_{i-1}} d(LS, p)$$

The most important advantage of this strategy is that it tries to cover the whole set of benign label structures in the training set as uniformly as possible [29].

*3.2.2. Constructing a vector representation of graphs*

Let $P = \{p_1, ..., p_M\}$ be the set of selected prototype label structures. We maintain in the memory for each graph $G$ a matrix $M_G$, where each element $d(i, j)$ represents the distance between the $i$-th label structure of the graph $G$ and the $j$-th prototype label structure.

$$M_G = \begin{pmatrix} d(1,1) & \dots & d(1,M) \\ \vdots & \ddots & \vdots \\ d(N,1) & \dots & d(N,M) \end{pmatrix} \tag{4}$$

We represent a graph $G$ by a vector $E_G$ containing $M$ coordinates, where the $j$-th coordinate is the sum of the weighted similarities between all the label structures of $G$ and the $j$-th prototype label structure.

$$E_G = (e_1, ..., e_M) \text{ where } e_j = \sum_{i=1}^{N} sim(i, j) w_i \tag{5}$$

In this Formula, $sim(i, j)$ represents the similarity between the label structure $LS_i$ of $G$ and the $j-$th prototype label structure. The similarity between two label structures $LS_1$ and $LS_2$ is computed with the edit distance $d$ between the two label structures (see Definition 2) which is normalised with the size of the label structures as follows:

$$sim(LS_1, LS_2) = 1 - \frac{d(LS_1, LS_2)}{1 + \max(|LS_1|, |LS_2|)} \tag{6}$$

$|LS_i|$ is the total number of edge labels in the label structure, i.e., the degree of the corresponding vertex computed as $|LS_i| = \sum_{t=1}^{\Sigma_E} L_i^-(t) + L_i^+(t)$.

$w_i$ is the weight of the label structure $LS_i$ computed as the total number of edge labels in the label structure $LS_i$ divided by 2 times the total number of edges in the graph $G$, i.e., $w_i = \frac{|LS_i|}{2|G|}$. The product $sim(i, j)w_i$ is called the impact of $LS_i$ on $E_G$ in regards to the $j$-th prototype label structure.

We compute the distance between two graphs $G_1$ and $G_2$ with the Euclidean distance ($L^2$ norm) between their characteristic vectors $E_{G_1}$ and $E_{G_2}$:

$$d_e(G_1, G_2) = L^2 norm(E_{G_1}, E_{G_2}) \tag{7}$$

As two similar graphs contain necessarily similar label structures, anomalous graphs will be far away from the prototype label structures. Unlike the cosine similarity used by StreamSpot [16] which considers only the number of common substructures between two graphs, LEADS compares implicitly the substructures of the two graphs by comparing them to the same prototypes. This allows to separate the anomalous graphs from the benign ones in the Euclidean space since all prototypes are benign and cover all the set of benign training label structures uniformly. Therefore the anomalous vectors will be far away from the benign ones.

We can also see that the distance between two graphs is proportional to the sum of the distances between their label structures. In fact:

$$d_e(G_1, G_2) = \left( \sum_{i=1}^{M} \Big( E_{G_1}(i) - E_{G_2}(i) \Big)^2 \right)^{\frac{1}{2}}$$

$$d_e(G_1, G_2) = \left( \sum_{i=1}^{M} \left( \sum_{j=1}^{N} sim(LS_j^{G_1}, p_i) w(LS_j^{G_1}) - \sum_{j=1}^{N} sim(LS_j^{G_2}, p_i) w(LS_j^{G_2}) \right)^2 \right)^{\frac{1}{2}}$$

$$\begin{aligned} d_e(G_1, G_2) = & \left( \sum_{i=1}^{M} \left( \sum_{j=1}^{N} \Big( sim(LS_j^{G_1}, p_i) w(LS_j^{G_1}) \right. \right. \\ & \left. \left. - sim(LS_j^{G_2}, p_i) w(LS_j^{G_2}) \Big) \right)^2 \right)^{\frac{1}{2}} \end{aligned}$$

$$\begin{aligned} d(G_1, G_2) = & \left( \sum_{i=1}^{M} \left( \sum_{j=1}^{N} \Big( w(LS_j^{G_1})(1 - \frac{d(LS_j^{G_1}, p_i)}{1 + \max(|LS_j^{G_1}|, |pi|)}) \right. \right. \\ & \left. \left. - w(LS_j^{G_2})(1 - \frac{d(LS_j^{G_2}, p_i)}{1 + \max(|LS_j^{G_2}|, |p_i|)}) \Big) \right)^2 \right)^{\frac{1}{2}} \end{aligned}$$

$$\begin{aligned} d_e(G_1, G_2) = & \left( \sum_{i=1}^{M} \left( \sum_{j=1}^{N} \Big( w(LS_j^{G_1}) - w(LS_j^{G_2}) - \frac{w(LS_j^{G_1}) d(LS_j^{G_1}, p_i)}{1 + \max(|LS_j^{G_1}|, |pi|)} \right. \right. \\ & \left. \left. + \frac{w(LS_j^{G_2}) d(LS_j^{G_2}, p_i)}{1 + \max(|LS_j^{G_2}|, |p_i|)} \Big) \right)^2 \right)^{\frac{1}{2}} \end{aligned}$$

$$\begin{aligned} d_e(G_1, G_2) = & \left( \sum_{i=1}^{M} \left( \sum_{j=1}^{N} \Big( \frac{|LS_j^{G_1}|}{2|G_1|} - \frac{|LS_j^{G_2}|}{2|G_2|} - \frac{|LS_j^{G_1}|}{2|G_1|} \frac{d(LS_j^{G_1}, p_i)}{1 + \max(|LS_j^{G_1}|, |pi|)} \right. \right. \\ & \left. \left. + \frac{|LS_j^{G_2}|}{2|G_2|} \frac{(LS_j^{G_2}, p_i)}{1 + \max(|LS_j^{G_2}|, |p_i|)} \Big) \right)^2 \right)^{\frac{1}{2}} \end{aligned}$$

as $\sum_{j=1}^{N} |LS_j^G| = 2|G|$, we have:

$$\begin{aligned} d_e(G_1, G_2) = & \left( \sum_{i=1}^{M} \left( \sum_{j=1}^{N} \Big( \frac{|LS_j^{G_2}| d(LS_j^{G_2}, p_i)}{2|G_2|(1 + \max(|LS_j^{G_2}|, |p_i|))} \right. \right. \\ & \left. \left. - \frac{|LS_j^{G_1}| d(LS_j^{G_1}, p_i)}{2|G_1|(1 + \max(|LS_j^{G_1}|, |pi|))} \Big) \right)^2 \right)^{\frac{1}{2}} \end{aligned}$$

Using Cauchy-Schwartz inequality, this gives:

$$d_e(G_1,G_2) \leq \Bigg( \sum_{i=1}^{M} \Bigg( \sum_{j=1}^{N} \frac{|LS_j^{G_2}|}{2|G_2|} \sum_{j=1}^{N} \frac{d(LS_j^{G_2},p_i)}{1+\max(|LS_j^{G_2}|,|p_i|)} - \sum_{j=1}^{N} \frac{|LS_j^{G_1}|}{2|G_1|} \sum_{j=1}^{N} \frac{d(LS_j^{G_1},p_i)}{1+\max(|LS_j^{G_1}|,|pi|)} \Bigg)^2 \Bigg)^{\frac{1}{2}}$$

So,

$$d_e(G_1,G_2) \leq \Bigg( \sum_{i=1}^{M} \Bigg( \sum_{j=1}^{N} \frac{d(LS_j^{G_2},p_i)}{1+\max(|LS_j^{G_2}|,|p_i|)} - \sum_{j=1}^{N} \frac{d(LS_j^{G_1},p_i)}{1+\max(|LS_j^{G_1}|,|pi|)} ) \Bigg)^2 \Bigg)^{\frac{1}{2}}$$

As $\max(|LS_j^G|,|p_i|) \geq 0$, we have:

$$d_e(G_1,G_2) \leq \sqrt{\sum_{i=1}^{M} \Bigg( \sum_{j=1}^{N} \Big( d(LS_j^{G_2},p_i) - d(LS_j^{G_1},p_i) \Big) \Bigg)^2}$$

As GED is a metric, $d$ satisfies the triangle inequality, i.e., $d(x,z) \leq d(x,y) + d(y,z)$. Thus:

$$d_e(G_1,G_2) \leq \sqrt{\sum_{i=1}^{M} \Bigg( \sum_{j=1}^{N} d(LS_j^{G_2},LS_j^{G_1} \Bigg)^2}$$

So, $d_e(G_1,G_2) \leq \sqrt{M} \Big( \sum_{j=1}^{N} d(LS_j^{G_2},LS_j^{G_1} \Big)$

That is, the upper bound of the Euclidean distance of a pair of graphs $G_1$ and $G_2$ is proportional to the sum of the differences between the edit distance of their label structures. Given $d$ is a metric, the more similar are the label structures, the smaller will be their edit distance, and consequently, the smaller will be the Euclidean distance between $G_1$ and $G_2$ in the vector space.

### *3.3. Anomaly detection in the graph Stream*

To detect anomalous graphs, we use clustering. The clusters are constructed on benign graphs in a training phase during which we deal only with benign graphs. Clustering is achieved on the vector representations of the graphs. Then, the incoming graphs are compared to the clusters to spot anomalous ones. We describe in the following the clustering process and the anomaly detection one.

*Cluster construction:.* We use a set of benign graphs that represent the normal behaviour of the system. We construct for each of these graphs a vector representation as described in Section 3.2.2. Then, we use the $k$-Medoid algorithm to organize them into $k$ clusters. $k$ is chosen in such way that it maximizes the silhouette coefficient [38] in order to well separate the clusters from each other. Then, an anomaly threshold is assigned to each cluster using Cantelli's

inequality [39]. These thresholds are set to 3 times the standard deviation (std) of the distances that are greater than the mean distance between the cluster's graphs and the Medoid [16]. Finally, we compute the center of each cluster, which is the average of the characteristic vectors of the graphs in the cluster.

*Anomaly Detection (Algorithm 1): .* the arrival of a new edge $e = (i, l_i, j, l_j, l_e, G)$ impacts two label structures $LS_i$ and $LS_j$ and vector $E_G$ of the graph $G$. For both LSs, we have two possible cases:

1. The label structure appears for the first time (lines 2 and 17): In this case, we compute its distances to all the prototype label structures, then we add the impact of the new label structure to vector $E_G$ (lines 2-8 and 17-23).
2. The label structure already exists. In this case, we update vector $E_G$ by subtracting the old impact of the label structure and then adding its new impact(lines 9-15 and 24-30).

To compute the new impacts of label structures $LS_i$ and $LS_j$ following the arrival of an edge directed from vertex $i$ to vertex $j$ and labeled $l_e$, we need to update their distances to the $M$ prototypes. This operation can be done incrementally using the the old distances stored in Matrix $M_G$. The new distance $d'(i, m)$ (resp. $d'(j, m)$) between $LS_i$ (resp. $LS_j$) and the $m^{th}$ prototype label structure is given by :
For $LS_i$ which has a new outgoing edge labeled $l_e$

$$d'(i,m) = \begin{cases} d(i,m)+1 & if\ (|L_i^+|+1 > |L_m^+|)\ and \\ & (L_i^+(l_e)+1 > L_m^+(l_e)) \\ d(i,m)-1 & if\ (|L_i^+|+1 \leq |L_m^+|)\ and \\ & (L_i^+(l_e)+1 \leq L_m^+(l_e)) \\ d(i,m) & Else \end{cases}$$

For $LS_j$ which has a new incoming edge labeled $l_e$:

$$d'(j,m) = \begin{cases} d(j,m)+1 & if\ (|L_j^-|+1 > |L_m^-|)\ and \\ & (L_j^-(l_e)+1 > L_m^-(l_e)) \\ d(j,m)-1 & if\ (|L_j^-|+1 \leq |L_m^-|)\ and \\ & (L_j^-(l_e)+1 \leq L_m^-(l_e)) \\ d(j,m) & Else \end{cases}$$

where $|L_i^-|$ (resp. $|L_i^+|$) is the number of incoming labels, i.e., in-degree (resp. the number of outgoing labels, i.e., out-degree) of the label structure $LS_i$.

After updating the characteristic vector $E_G$, we need to reclassify the graph $G$ (reexamine the graph $G$, since its characteristic vector has changed). To do this, we compute the Euclidean distance between $E_G$ and all the $k$ centers of benign graph clusters to find the closest cluster to $G$ (line 32). This operation takes $\mathcal{O}(kM)$. Then, we reclassify the graph $G$ as follows: Let $C_n$ be the nearest cluster to $E_G$, if the distance between $E_G$ and $C_n$ is greater than the threshold assigned to $C_n$ then the graph is anomalous (line 33-34). Else $G$ is a benign graph since it can be assigned to the cluster $C_n$ (line 35-36)."

*3.4. Space and Time Complexity*

- **Space complexity**: For each graph $G$: the space occupied by its label structures is $\mathcal{O}(|V|(2|\Sigma_E|))$, the space occupied by its distance matrix $M_G$ is $\mathcal{O}(|V|M)$, and its characteristic vector $E_G$ occupies $\mathcal{O}(M)$ memory units. This leads to a total space complexity of $\mathcal{O}(|V|(M + 2|\Sigma_E|))$ per graph.

- **Computational complexity:** at the arrival of a new edge, updating the affected label structures is constant time, updating the distance between label structures takes $O(M)$, updating $E_G$ takes $\mathcal{O}(M)$ and finally the classification of the graph takes $\mathcal{O}(Mk)$ where $k$ is the number of clusters and $M$ is the number of prototypes. Therefore, the total time complexity is $\mathcal{O}(Mk)$ per edge.

## 4. Experimental results

In this section, we present the experiments we performed to evaluate our approach. All the experiments are performed on Intel i7 8700K (3.7 GHz) with 32 GB RAM using C++. We first describe the datasets. We present the performance metrics used for experiments. Then, we provide empirical study on LEADS and comparisons with StreamSpot [16].

*4.1. Datasets*

We used a cyber-security related dataset [4] [16] where anomalies are cyber-attacks. The datasets comprise 7 types of flow-graphs derived from 2 malicious and 5 normal activities. The characteristics of the 7 types of graphs are presented in Table 1. The malicious activities consist of the drive-by download attack triggered by visiting a malicious web address and a Java attack. The normal or benign activities involve ordinary internet browsing activity, like watching YouTube, downloading files, browsing news, checking Gmail, and playing a video game. The graphs of benign activities were compiled into 4 sub-datasets:

- **ALL**: comprises all activities

- **GFC**: comprises checking Gmail, playing video games and browsing news activities.

- **YDC**: comprises watching YouTube, Downloading and browsing news activities.

- **YDG**: comprises watching YouTube, Downloading and checking Gmail activities.

[4] https://github.com/cmuxstream/cmuxstream-data/blob/master/evolving/streamspot-raw.gz

**Data:** Stream of labeled directed edges, clusters of benign graphs of the training phase, set of prototype label structures $P$
**Result:** Detection of anomalous graphs

**while** *a new edge* $e =< i, l_s, j, l_d, l_e, G >$ *arrives* **do**
  // Update the vector $E_G$ ;
  **if** $LS_i$ *doesn't exists* **then** // a new LS appears
    $L_i^+(l_e) \leftarrow 1$;
    $\ell_i \leftarrow l_s$;
    **for** $m \leftarrow 1 \quad M$ **do**
      compute $d(LS_i, p_m)$;
      $E_G(m) \leftarrow \frac{E_G(m)*2(|G|-1)+sim(LS_i,p_m)}{2*|G|}$;
    **end**
  **else** // the label structure $LS_i$ already exists
    **for** $m \leftarrow 1 \quad M$ **do**
      $E_G(m) \leftarrow E_G(m) * 2(|G|-1) - sim(LS_i, p_m)) * |LS_i|$;
      update $M_G(i, m)$; // i.e., $d(LS_i, p_m)$
      $E_G(m) \leftarrow \frac{E_G(m)+(sim(LS_i,p_m))*(|LS_i|+1)}{2*(|G|+1)}$;
    **end**
    $L_i^+(l_e) \leftarrow L_i^+(l_e) + 1$; // Update $LS_i$
  **end**
  **if** $LS_j$ *doesn't exists* **then** // a new label structure appears
    $L_j^-(l_e) \leftarrow 1$;
    $\ell_j \leftarrow l_d$;
    **for** $m \leftarrow 1 \quad M$ **do**
      compute $d(LS_j, p_m)$;
      $E_G(m) \leftarrow \frac{E_G(m)*2(|G|-1)+sim(LS_j,p_m)}{2*|G|}$;
    **end**
  **else** // the label structure $LS_j$ already exists
    **for** $m \leftarrow 1 \quad M$ **do**
      $E_G(m) \leftarrow E_G(m) * 2(|G|-1) - (sim(LS_j, p_m)) * |LS_j|$;
      update $M_G(j, m)$; // i.e., $d(LS_j, p_m)$
      $E_G(m) \leftarrow \frac{E_G(m)+(sim(LS_j,p_m))*(|LS_j|+1)}{2*(|G|+1)}$;
    **end**
    $L_j^-(l_e) \leftarrow L_j^-(l_e) + 1$; // update $LS_j$
  **end**
  // Classification of the graph $G$
  compute the smallest distance between the graph $G$ and the centers of the $k$ clusters ;
  **if** *distance* $>$ *threshold of the nearest cluster* **then**
    the graph $G$ is anomalous ;
  **else**
    the graph $G$ is normal ;
  **end**
**end**

**Algorithm 1:** Pseudo-code of the anomaly detection in the streaming scenario

### *4.2. Performance metrics*

We use the following performance metrics to evaluate our approach, considering Malicious as the positive class: False Positive rate ($FPR$), False Negative rate ($FNR$), F1 score ($F1$), Balanced accuracy ($BACC$), Average Precision ($AP$), Area Under Curve ($AUC$). In all experiments, the performance metrics are computed on the instantaneous anomaly detection every 10,000 edges.

| Activity | #Graphs | Avg $\|V\|$ | Avg $\|E\|$ |
|---|---|---|---|
| YouTube | 100 | 8291.79 | 113228 |
| Download | 100 | 6827.3 | 37382 |
| Browsing news | 100 | 8637.34 | 112957 |
| Drive-by download (Attack) | 100 | 8890.8 | 28423 |
| Checking mail | 100 | 8831.46 | 310813 |
| Playing video games | 100 | 8990.09 | 294903 |
| Java (Attack) | 100 | 6881.37 | 27844 |

Table 1: Characteristics of the dataset's graphs

### *4.3. Empirical Study of LEADS*

#### *4.3.1. Experiment Settings*

This first set of experiments aims to calibrate the number of prototype label structures $M$. For this, we studied the behavior of our approach, by setting a range of values for the parameter $M = \{25, 50, 100\}$. For each value of $M$ we carried-out an experiment on the ALL sub-dataset using $\tau = 75\%$ of benign graphs selected randomly for the training. The remaining 25% of benign graphs (i.e., 125 benign graphs) with all attack graphs (200 attack graphs) constitute the set of test graphs which was divided in groups of $P = 50$ graphs.

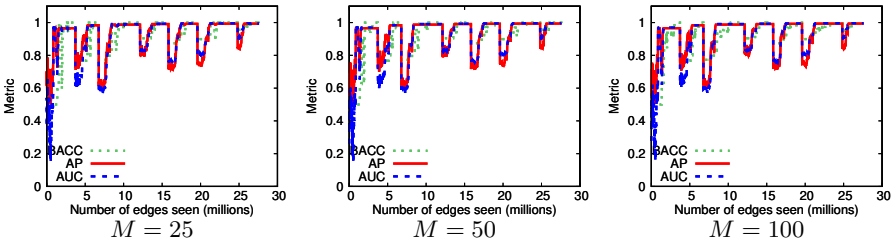
#### *4.3.2. Results*

Figure 3 shows the performance of LEADS on different values of $M$. We notice that the three curves are almost similar with periodic dips in each one. Each dip corresponds to the arrival of a new group of graphs. These drops of performance are justified by the fact that at the beginning of each period only a small portion of the global structures of the new graphs is known. The performance recovers as graphs start to grow to achieve rates $> 95\%$ for all metrics at the end of each period.

Table 2 gives the values of each performance metric at the end of the stream as well as the number of processed edges per second for each value of $M$. We can see that we obtain identical results regarding the anomaly detection efficiency for the 3 values of $M$ with a slightly better performance for $M = 100$. However, the run-times are not the same. It is obvious that the smaller $M$ is, the greater is the number of edges processed per second. We deduce from Figure 3 and Table 2 that the best value of $M$ for this benchmark is 25, because the detection for $M = 25$ is 3 times faster than the detection using 100 prototypes, and the differences in performance are not noticeable (0.003 in AUC) between the three values of $M$. Therefore, in all experiments that follow we set the value of $M$ to 25.

### *4.4. Comparative Study*

In this set of experiments, we present the results of the comparison of LEADS with StreamSpot [16].




$M = 25$ $M = 50$ $M = 100$

Figure 3: The effect of parameter $M$ on the performance of LEADS

| $M$ | BACC | F1 score | AP | AUC | #edges/sec |
|---|---|---|---|---|---|
| 25 | 0.996 | 0.997 | 0.992 | 0.993 | 92431 |
| 50 | 0.996 | 0.997 | 0.993 | 0.995 | 50233 |
| 100 | 0.996 | 0.997 | 0.994 | 0.996 | 26038 |

Table 2: The performance and run-time with different values of $M$

*4.4.1. Experiment Settings*

We conducted our tests on the 4 sub-datasets ALL, GFC, YDC and YDG using different rates of training graphs $\tau = 25\%, 50\%, 75\%$. The number of test graphs and test edges used in each experiment are given in Table 3. For a fair comparison, we run StreamSpot using the best combination of parameters [16]: sketch size $L = 1000$, chunk-size $C = 25, 100, 50$ respectively for YDC, GFC and ALL and $C = 50$ for the new sub-dataset YDG without any limit on the size of the cache $N = 100\%$ of the size of the stream for all sub-datasets.

| sub-dataset | $\tau$ | #Test Graph | | #Test edges(Millions) |
|---|---|---|---|---|
| | | #Benign graphs | #Anomalous Graphs | |
| ALL | 75% | 125 | 200 | 27.62 |
| | 50% | 250 | 200 | 49.95 |
| | 25% | 375 | 200 | 71.76 |
| GFC | 75% | 75 | 200 | 16.91 |
| | 50% | 150 | 200 | 28.14 |
| | 25% | 225 | 200 | 39.5 |
| YDC | 75% | 75 | 200 | 23.71 |
| | 50% | 150 | 200 | 42.15 |
| | 25% | 225 | 200 | 60.35 |
| YDG | 75% | 75 | 200 | 17.3 |
| | 50% | 150 | 200 | 29.35 |
| | 25% | 225 | 200 | 40.71 |

Table 3: Test experiments characteristics

*4.4.2. Results*

Figures 4, 5, 6 and 7 show the performances of the two methods for the 3 training rates 25%, 50% and 75% on the 4 datasets: ALL, GFC, YDC and YDG, respectively. We can see clearly that:

1. **Figures 4, 5, 6 and 7:** We notice a clear drop in performance (the periodic dips) at the arrival of a new group of $P$ graphs in all the curves. This

is due to the fact that only a small portion of the new coming test graphs are known at the beginning of each period. Therefore, at this stage the new benign graphs are misclassified as malicious ($FPR$ increase). Since they are dissimilar to the benign training graphs which are of complete size. Then, the performance recovers as soon as the new graphs start growing (FPR decrease).

2. **Figure 4 (dataset ALL):** This figure shows that both approaches behave very well and almost perfectly at the end of each period in terms of $AP$ and $AUC$ ($AP > 95\%$ and $AUC > 95\%$) for the 3 training rates. Furthermore, our approach clearly outperforms StreamSpot in term of $BACC$ for the 3 values of $\tau$. We notice also that the impact of the parameter $\tau$ on Streamspot is considerable. Indeed, the value of $BACC$ changes significantly by varying the training rate.

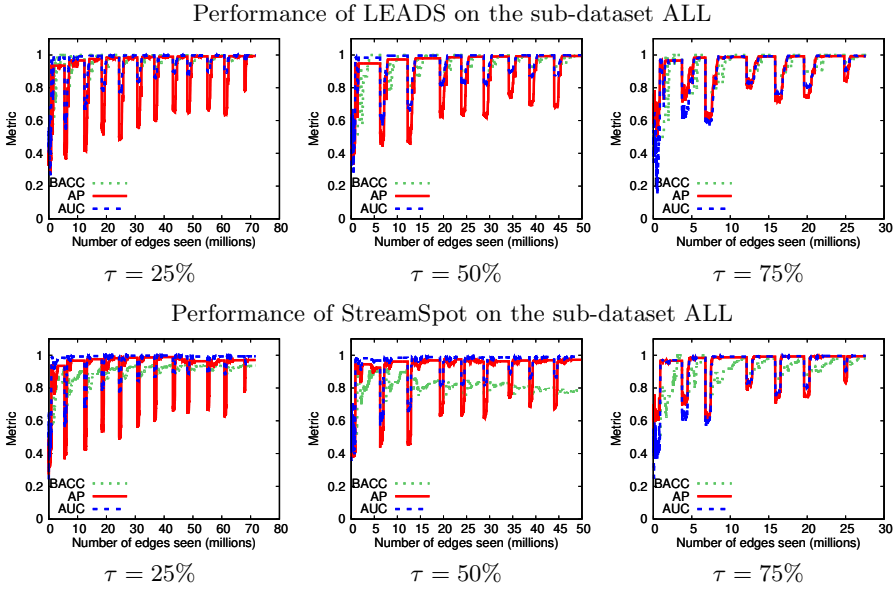



Figure 4: Comparison of LEADS with StreamSpot on the sub-dataset ALL (P=50)

3. **Figure 5 (dataset GFC):** The results of the two approaches are very satisfactory with slightly shorter detection delays in Streamspot. The performance of both approaches remains very high even for a smaller rate of training $\tau = 25\%$.
4. **Figure 6 (dataset YDC):** We notice that the performance of our approach are better than Streamspot's performances with higher $AP$ and $BACC$ and shorter anomaly detection delays. The superiority of LEADS over StreamSpot is more visible for smaller rates of training especially for the $BACC$ metric. For $\tau = 25\%$ we see that the performances of both approaches drop, however our results remain very satisfactory with rates $> 90\%$ in all metrics while the results of StreamSpot are under $90\%$

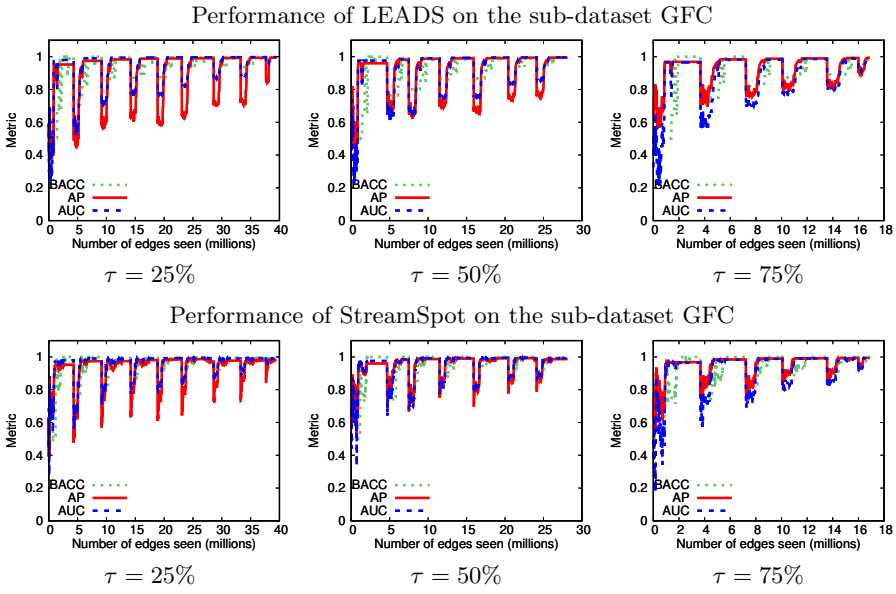



Figure 5: Comparison of LEADS with StreamSpot on the sub-dataset GFC (P=50)

especially for the $BACC$ metric.

5. **Figure 7 (dataset YDG):** The $AP$ and $AUC$ of the two approaches are very high and satisfactory even for the smaller rate of training $\tau = 25\%$. In addition, our approach clearly outperforms Streamspot in terms of $BACC$. The superiority of LEADS is very clear for the greater rates of training.

Table 4 presents the performance of the detection of both approaches (StreamSpot (STRSP) [16] and our approach LEADS) at the end of the stream as well as the number of processed edges per second. Our results are very satisfactory with null false negative rates in all experiments and relatively small false positive rates. In addition, we see clearly that LEADS behaves almost perfectly in terms of $AP$ and $AUC$ with values above 99% in all scenarios. Moreover, LEADS outperforms StreamSpot in the majority of experiments expect for two scenarios (GFC 75% and GFC 50%) where the two approaches are equivalent. LEADS consistently outperforms StreamSpot for the $BACC$ and $F1$ score metrics and especially for the small rates of training. Concerning the speed of both approaches, we can see that our approach is up to 3 times faster than StreamSpot for the ALL and the YDC instances and up to 6 times faster for the GFC instances.

In summary, these results demonstrate the effectiveness of LEADS compared to StreamSpot.

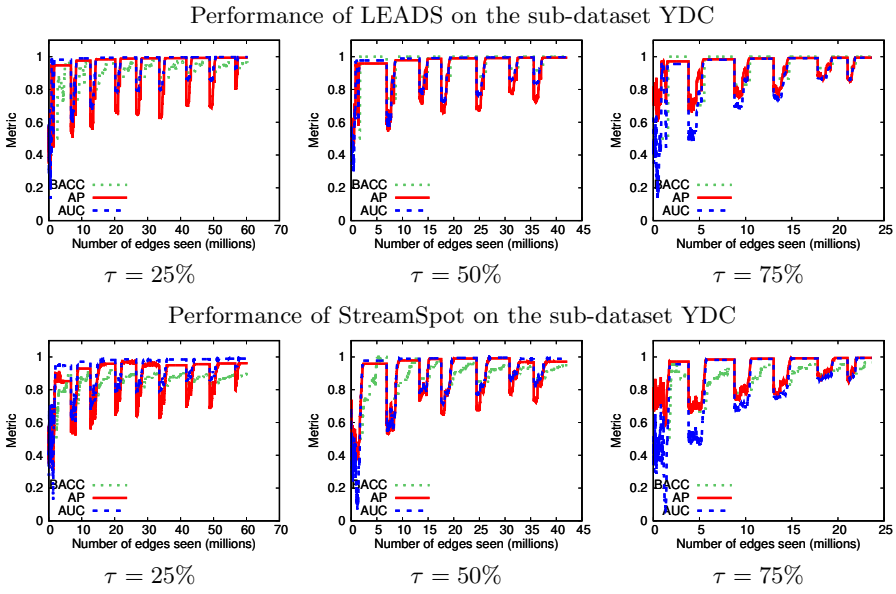



Figure 6: Comparison of LEADS with StreamSpot on the sub-dataset YDC (P=50)

## 5. Conclusion

This paper presents a new approach for detecting anomalies in a stream of heterogeneous graphs. We have introduced a new representation of graphs by vectors. This representation is based on weighted sub-structures and graph edit distance. It enables a rapid update of the vectors of graphs and consumes a limited amount of memory space per graph. The results obtained show the effectiveness of our approach, which achieves over 95% precision while processing more than $90,000$ edges per second.

## References

[1] S. Lagraa, J. François, A. Lahmadi, M. Miner, C. A. Hammerschmidt, R. State, Botgm: Unsupervised graph mining to detect botnets in traffic flows, in: 1st Cyber Security in Networking Conference, CSNet 2017, Rio de Janeiro, Brazil, October 18-20, 2017, 2017, pp. 1–8.

[2] L. Akoglu, H. Tong, D. Koutra, Graph based anomaly detection and description: A survey, Data Min. Knowl. Discov. 29 (3) (2015) 626–688.

[3] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, N. F. Samatova, Anomaly detection in dynamic networks: a survey, Wiley Interdisciplinary Reviews: Computational Statistics 7 (3) (2015) 223–247.

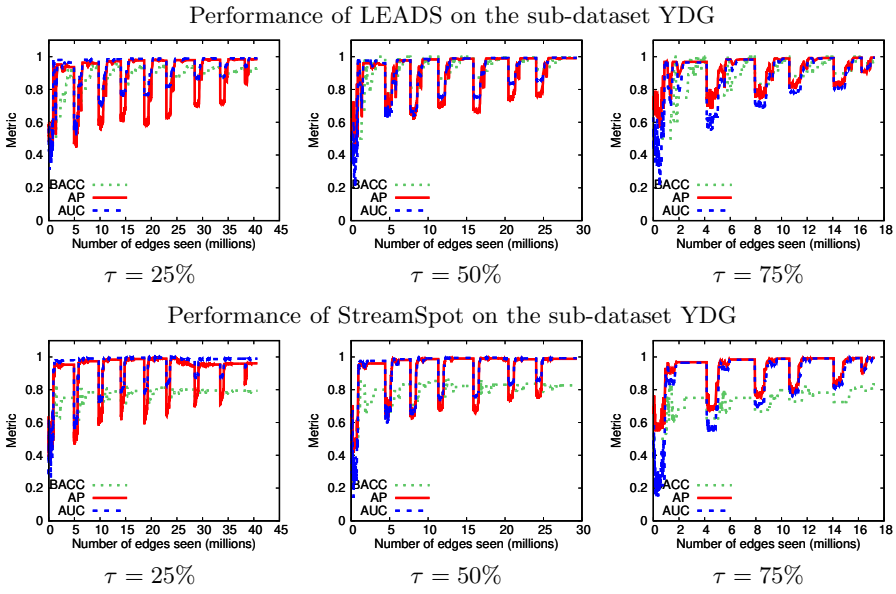



Figure 7: Comparison of LEADS with StreamSpot on the sub-dataset YDG (P=50)

[4] C. C. Noble, D. J. Cook, Graph-based anomaly detection, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03, 2003, pp. 631–636.

[5] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, J. Han, On community outliers and their efficient detection in information networks, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10, 2010, pp. 813–822.

[6] A. Leman, M. Mary, F. Christos, oddball: Spotting anomalies in weighted graphs, in: Advances in Knowledge Discovery and Data Mining, 2010, pp. 410–421.

[7] P. E. E., F. Christos, S. N. D., Parcube: Sparse parallelizable tensor decompositions, in: Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, 2012, pp. 521–536.

[8] H. Sun, J. Huang, J. Han, H. Deng, P. Zhao, B. Feng, gskeletonclu: Density-based network clustering via structure-connected tree division or agglomeration, in: Data Mining (ICDM), 2010 IEEE 10th International Conference on, 2010, pp. 481–490.

[9] C. C. Aggarwal, K. Subbian, Event detection in social streams, in: Proceedings of the 2012 SIAM international conference on data mining, 2012, pp. 624–635.

| | $\tau$ | method | BACC | F1 | AP | AUC | FPR | FNR | #edges(s) |
|---|---|---|---|---|---|---|---|---|---|
| ALL | 25% | STRSP | 0.937 | 0.894 | 0.970 | 0.993 | 0.125 | **0** | 26014 |
| | | LEADS | **0.997** | **0.995** | **0.993** | **0.998** | **0.005** | **0** | **94705** |
| | 50% | STRSP | 0.782 | 0.735 | 0.971 | 0.990 | 0.060 | 0.375 | 25360 |
| | | LEADS | **0.996** | **0.995** | **0.993** | **0.998** | **0.008** | **0** | **69817** |
| | 75% | STRSP | 0.980 | 0.987 | **0.994** | **0.996** | 0.040 | **0** | 25836 |
| | | LEADS | **0.996** | **0.997** | 0.992 | 0.993 | **0.008** | **0** | **92431** |
| GFC | 25% | STRSP | 0.991 | 0.990 | 0.980 | 0.991 | 0.017 | **0** | 15374 |
| | | LEADS | **0.995** | **0.995** | **0.993** | **0.997** | **0.008** | **0** | **96844** |
| | 50% | STRSP | **0.996** | **0.997** | 0.986 | 0.991 | **0.006** | **0** | 15215 |
| | | LEADS | **0.996** | **0.997** | **0.993** | **0.996** | **0.006** | **0** | **94676** |
| | 75% | STRSP | **1** | **1** | **0.994** | **0.993** | **0** | **0** | 15618 |
| | | LEADS | 0.993 | 0.997 | 0.992 | 0.989 | 0.013 | **0** | **96179** |
| YDC | 25% | STRSP | 0.895 | 0.894 | 0.962 | 0.989 | 0.208 | **0** | 26045 |
| | | LEADS | **0.964** | **0.961** | **0.993** | **0.997** | **0.071** | **0** | **80956** |
| | 50% | STRSP | 0.95 | 0.963 | 0.970 | 0.988 | 0.1 | **0** | 25999 |
| | | LEADS | **1** | **1** | **0.993** | **0.996** | **0** | **0** | **97972** |
| | 75% | STRSP | 0.966 | 0.987 | **0.994** | **0.993** | 0.066 | **0** | 27098 |
| | | LEADS | **1** | **1** | **0.994** | **0.993** | **0** | **0** | **96052** |
| YDG | 25% | STRSP | 0.793 | 0.811 | 0.961 | 0.989 | 0.413 | **0** | 27298 |
| | | LEADS | **0.926** | **0.923** | **0.984** | **0.990** | **0.146** | **0** | **89750** |
| | 50% | STRSP | 0.826 | 0.884 | 0.988 | **0.992** | 0.346 | **0** | 22223 |
| | | LEADS | **0.996** | **0.997** | **0.990** | **0.992** | **0.006** | **0** | **96790** |
| | 75% | STRSP | 0.833 | 0.941 | **0.994** | **0.993** | 0.333 | **0** | 25334 |
| | | LEADS | **0.993** | **0.997** | **0.994** | **0.993** | **0.013** | **0** | **105762** |

Table 4: Performance results at the end of the stream. A comparison between StreamSpot (STRSP) [16] and our approach LEADS.


[10] J. Sun, C. Faloutsos, C. Faloutsos, S. Papadimitriou, P. S. Yu, Graphscope: Parameter-free mining of large time-evolving graphs, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07, 2007, pp. 687–696.

[11] C. Aggarwal, K. Subbian, Evolutionary network analysis: A survey, ACM Comput. Surv. (2014) 10:1–10:36.

[12] H. Bunke, P. Dickinson, A. Humm, C. Irniger, M. Kraetzl, Computer network monitoring and abnormal event detection using graph matching and multidimensional scaling, in: Proceedings of the 6th Industrial Conference on Data Mining Conference on Advances in Data Mining: Applications in Medicine, Web Mining, Marketing, Image and Signal Mining, ICDM'06, 2006, pp. 576–590.

[13] D. Koutra, N. Shah, J. T. Vogelstein, B. Gallagher, C. Faloutsos, Deltacon: Principled massive-graph similarity function with attribution, ACM Trans. Knowl. Discov. Data 10 (2016) 28:1–28:43.

[14] C. C. Aggarwal, Y. Zhao, P. S. Yu, Outlier detection in graph streams, in: 2011 IEEE 27th International Conference on Data Engineering, 2011, pp. 399–409.

[15] S. Ranshous, S. Harenberg, K. Sharma, N. F. Samatova, A scalable approach for outlier detection in edge streams using sketch-based approximations, in: Proceedings of the 2016 SIAM International Conference on Data Mining, Miami, Florida, USA, May 5-7, 2016, 2016, pp. 189–197.

[16] E. Manzoor, S. M. Milajerdi, L. Akoglu, Fast memory-efficient anomaly detection in streaming heterogeneous graphs, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, 2016, pp. 1035–1044.

[17] H. Bunke, G. Allerman, Inexact graph matching for structural pattern recognition, Pattern Recognition Letters - PRL 1 (4) (1983) 245–253.

[18] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International journal of pattern recognition and artificial intelligence 18 (03) (2004) 265–298.

[19] R. Kaspar, F. Miquel, F. Andreas, B. Horst, Approximation of graph edit distance in quadratic time, in: Graph-Based Representations in Pattern Recognition, Springer International Publishing, 2015, pp. 3–12.

[20] A. Sanfeliu, K.-S. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE transactions on systems, man, and cybernetics (3) (1983) 353–362.

[21] Z. Zeng, A. K. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: On approximating graph edit distance, Proceedings of the VLDB Endowment 2 (1) (2009) 25–36.

[22] G. Wang, B. Wang, X. Yang, G. Yu, Efficiently indexing large sparse graphs for similarity search, IEEE Transactions on Knowledge and Data Engineering 24 (3) (2012) 440–451.

[23] X. Zhao, C. Xiao, X. Lin, W. Wang, Efficient graph similarity joins with edit distance constraints, in: Data Engineering (ICDE), 2012 IEEE 28th International Conference on, 2012, pp. 834–845.

[24] W. Zheng, L. Zou, X. Lian, D. Wang, D. Zhao, Efficient graph similarity search over large graph databases, IEEE Transactions on Knowledge and Data Engineering 27 (4) (2015) 964–978.

[25] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Anal. Appl. 13 (1) (2010) 113–129.

[26] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years, International Journal of Pattern Recognition and Artificial Intelligence (2014).

[27] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, K. M. Borgwardt, Weisfeiler-lehman graph kernels, Journal of Machine Learning Research (2011) 2539–2561.

[28] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1365–1374.

[29] K. Riesen, H. Bunke, Graph classification based on vector space embedding, International Journal of Pattern Recognition and Artificial Intelligence 23 (2009) 1053–1081.

[30] G. A. Mills-Tettey, A. Stentz, M. B. Dias, The dynamic hungarian algorithm for the assignment problem with changing costs, Tech. Rep. CMU-RI-TR-07-27, Robotics Institute, Carnegie Mellon University (2007).

[31] I. H. Toroslu, G. Üçoluk, Incremental assignment problem, Information Sciences 177 (2007) 1523–1529.

[32] O. Kostakis, Classy: fast clustering streams of call-graphs, Data Mining and Knowledge Discovery 28 (5) (2014) 1554–1585.

[33] C. C. Aggarwal, Data Streams: Models and Algorithms (Advances in Database Systems), Springer-Verlag, 2006.

[34] D. Eswaran, C. Faloutsos, S. Guha, N. Mishra, Spotlight: Detecting anomalies in streaming graphs, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '18, 2018, pp. 1378–1386.

[35] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, Journal of Algorithms 55 (1) (2005) 58 – 75.

[36] M. S. Charikar, Similarity estimation techniques from rounding algorithms, in: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, 2002, pp. 380–388.

[37] D. Eswaran, C. Faloutsos, Sedanspot: Detecting anomalies in edge streams, in: 2018 IEEE International Conference on Data Mining (ICDM), 2018.

[38] P. J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, Journal of Computational and Applied Mathematics 20 (1987) 53 – 65.

[39] G. Grimmett, D. Stirzaker, Probability and random processes, Oxford university press, 2001.