SNAPSKETCH: Graph Representation Approach for Intrusion Detection in a Streaming Graph

Ramesh Paudel
rpaudel42@students.tntech.edu
Tennessee Technological University
Cookeville, TN

William Eberle
weberle@tntech.edu
Tennessee Technological University
Cookeville, TN

ABSTRACT

In this paper, we propose a novel unsupervised graph representation approach in a graph stream called SnapSketch that can be used for anomaly detection. It first performs a fixed-length random walk from each node in a network and constructs n-shingles from a walk path. The top discriminative n-shingles identified using a frequency measure are projected into a dimensional projection vector chosen uniformly at random. Finally, a graph is sketched into a low-dimensional sketch vector using a simplified hashing of the projection vector and the cost of shingles. Using the learned sketch vector, anomaly detection is done using the state-of-the-art anomaly detection approach called RRCF [1]. SnapSketch has several advantages, including fully unsupervised learning, constant memory space usage, entire-graph embedding, and real-time anomaly detection.

CCS CONCEPTS

• **Computing methodologies** → *Anomaly detection*; • **Networks** → *Denial-of-service attacks.*

KEYWORDS

graph sketching, anomaly detection, graph stream

ACM Reference Format:
Ramesh Paudel and William Eberle. 2020. SNAPSKETCH: Graph Representation Approach for Intrusion Detection in a Streaming Graph. In *MLG '20: 16th International Workshop on Mining and Learning with Graphs, Aug 23-27, 2020, San Diego, California.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Learning problems like classification, link prediction, community detection, role discovery, anomaly detection, visualization, etc., in graphs can be accomplished using standard machine learning algorithms. However, the primary challenge is in the extraction of informative, discriminating, and independent features that can represent or encode a graph structure so that it can be used as input to a machine learning model. For example, to predict the links between two individuals in a social network, we might want to extract pairwise information that represents the relationship strength between two individuals, like the number of common friends. Or to classify a node, we might want to extract node properties like a local neighborhood, global position in the graph, etc. Since there is no common or simple way to encode or represent these high dimensional graphs, the feature extraction in graphs is challenging. One way is to map user-defined statistics, like degree, label, or neighborhood statistics, into a feature vector. The disadvantage of these kinds of approaches is that the feature vector transformation may lead to a loss in the topological information, feature generation is time-consuming and expensive, and features generated beforehand cannot adapt during the learning process [16]. On the other hand, a classical feature representation technique like a graph kernel can evaluate the similarity between two graphs G and G' by recursively decomposing them into atomic substructures through random walks [12, 21], shortest paths [6], subgraphs [34], subtrees [20, 32], etc., and define a similarity function over the substructures (e.g., counting the number of common substructures across G and G'). However, most of these atomic substructures in graphs are extracted using a well-defined function and when used on large datasets of graphs, it leads to building very high dimensional, sparse, and non-smooth representations yielding poor generalization [16, 41].

Recently, a graph representation technique called graph embedding has gained popularity. Graph embedding is an approach that transforms nodes, edges, subgraphs, graphs and their features into a low dimensional vector space whilst optimally preserving their properties. Embedding techniques are categorized into factorization-based [5, 29], random walk-based [14, 31], and deep learning-based [22, 40] techniques. Though graph embedding shows promising results among graph representation methods, the computation cost is usually high. Also, most of the research on graph embedding is focused on a node and edge embedding and little work has been done for embedding the subgraphs or the complete graphs. Furthermore, there is limited research when it comes to embedding a streaming graph.

In streaming graphs, sketching techniques have been used as a part of an efficient learning task [11, 19, 26]. Sketching is another graph representation technique that projects the higher dimensional object like graphs into a lower-dimensional feature vector. In other words, sketching summarizes big and streaming graphs while preserving their original properties like graph distances, cut, and density. These approaches can handle streaming, heterogeneous graphs, with low space overhead and real-time processing of the graph [3, 26]. In this work, we propose an unsupervised general-purpose graph sketching technique called SnapSketch that

MLG '20, Aug 23-27, 2020 , San Diego, California

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

can represent a snapshot of a graph stream at any time t into a fixed-size sketch vector. SnapSketch first performs a fixed-length random walk from each node in a graph and constructs n-shingles from a walk path. The top discriminative n-shingles are identified using a frequency measure and are projected into a $d-$dimensional projection vector $h_{[1,...,d]} = \{1, 0\}$ chosen uniformly at random with probability r. Finally, the graph is sketched into a low dimensional sketch vector using a simplified hashing of the projection vector and the cost of shingles. As a result, the obtained sketch vector can then be processed by standard machine learning algorithms for anomaly detection.

The contributions of this work can be summarized as follows:

- We propose SnapSketch , an unsupervised graph representation approach on a graph stream.
- We demonstrate the effectiveness of SnapSketch on anomalous hotspot detection in graph streams.
- SnapSketch outperforms baseline graph stream anomaly detection approaches on several large real-world graph streams.

2 RELATED WORK

Our proposed graph representation technique maps graph-structured data into a fixed-size sketch vector. This work closely resembles three popular techniques of graph representation: graph kernel, graph embeddings, and graph sketching.

A graph kernel recursively decomposes the graph into atomic substructure like graphlet, subtree, random walk, etc., and represents a graph structure as a vector containing the counts of the atomic substructures. The similarity of the two graphs can be measured using the inner product of such vectors. Kondor and Lafferty [23] first proposed the idea of constructing kernels (called a diffusion kernel) between the nodes of a single graph and it was later extended by Smola and Kondor [37]. Graph kernels based on walks and paths [12, 18, 21, 39], subgraph (or graphlet) [20, 34, 35], and subtree [10, 32] are proposed by various researchers. Furthermore, other graph kernels like Deep Divergence Graph Kernels (DDGK) [4] measure the similarity between a pair of graphs by the divergence in their structures. However, while graph kernels are defined using a fixed set of substructures or patterns, their capacity to distinguish graphs of different classes does not adapt to the given data distribution [24].

Like graph kernels, a graph embedding represents the graph-structured data into a vector space. Initially, a graph embedding evolved as a dimensionality reduction technique where a graph with $D-$dimensions is embedded into a $d-$dimensional vector space, where $d << D$. Recently, the research has shifted towards generating a better representation of graphs. Factorization-based graph embedding represents a graph in the form of a matrix (e.g., node adjacency matrix, Laplacian matrix, node transition probability matrix, etc.) and factorizes this matrix to obtain a node embedding. GraRep [8] and HOPE [29] are examples of factorization-based embedding. The problem with factorization-based methods is that they are not capable of learning an arbitrary function, e.g., to explain network connectivity. Thus, unless explicitly included in their objective function, they cannot learn structural equivalences [13]. Embedding techniques like DeepWalk [31], node2vec [14], HARP [9], etc., are based on a random walk. But, the shallow models based on random walk or a factorization method cannot capture complex and highly non-linear network structures, resulting in sub-optimal network representations [40]. However, deep learning-based methods like Structural Deep Network Embedding (SDNE) [40] and Graph Convolution Networks (GCNs) [22] can effectively capture the highly non-linear network structure and preserve the global and local structure of the network.

The embedding techniques discussed above focus on node embeddings. The application of such an embedding is limited to node classification and edge prediction. Approaches like PATCHY-SAN [28], graph2vec [27], Sub2vec [2], GAT2VEC [33], etc., propose the embedding of the complete graph and are better suited for graph classification. In a dynamic graph, an embedding technique like GCN [22] is proposed. However, GCN [22] does not consider the timing factors, which cannot be ignored in the dynamic graphs. NetWalk [42] learns a network representation in a dynamic network by encoding the vertices of the dynamic network to vector representations by a clique embedding that can be updated dynamically as the network evolves. AddGraph [44] uses an extended temporal GCN with an attention model that can capture both long-term patterns and the short-term patterns in dynamic graphs. The embedding techniques are based on a neural network. Due to stacking multiple layers of graph convolution or high complexity of backpropagation, they do not scale well for large graphs. Also, the majority of graph embedding techniques cannot handle dynamic and heterogeneous graphs.

Sketching is a useful graph representation technique in a streaming setting for summarizing graphs that can be implicitly stored in a small space and constructed when required. Using sketching, the summary of the entire graph can be computed without storing the whole graph. gSketch [43], SpotLight [11], StreamSpot [26], TCM [38], NetCondense [1], SBG-Sketch [19], GODIT [30], etc., are examples of sketching techniques in streaming graphs. In this work, we propose a sketching approach in a graph stream called SnapSketch that uses a simplified hashing of the discriminative shingles generated from a biased-random walk. The sketches generated by SnapSketch can then be used for anomaly detection in dynamic graphs.

3 PROPOSED METHODOLOGY

3.1 Preliminaries

In this section, we formally define the problem and notations, and introduce our streaming graph model.

3.1.1 Problem Setting. Let $G_s = \{G_1, G_2, ..., G_t, ...\}$ be a graph stream where each G_t denotes a graph at time t. We consider a graph $G_t = (v, e, f)$ as a generic weighted (un)directed heterogeneous graph. Whenever a new graph G_t arrives in the stream, a biased-random walk of a fixed length l is performed from each node in G_t extracting a walk path $p_{G_t} = \{v_1, v_2, ..., v_l\}$. The n-shingles are then constructed from a walk path p_{G_t}.

Definition 3.1. A **shingle** is a contiguous sub-sequence contained in a walk path p_{G_t}.

Definition 3.2. The **n-shingle** $S(v, n)$ is a sub-sequence of size n constructed from a random walk path p_{G_t}.

The value of n can be randomly chosen to be any constant. However, the lower value of n might not represent any significant neighborhood information. For example, if an $n = 2$ shingle represents the edge, it might not provide enough information about the neighborhood beyond one hop (or one neighbor). On the other hand, a larger value of n will produce a more expressive local neighborhood [26]. From the set of shingles (S_t) present in a graph G_t, the top k discriminative shingles (S_t^k) are determined, i.e., shingles with the highest frequency values. Each of the top k discriminative shingles are then projected into a $d-$dimensional vector $h_{[1,...,d]} = \{1, 0\}$ chosen uniformly at random with probability r.

Definition 3.3. **Sketching** is the data representation technique where the higher dimensional graph object G_t can be projected into a lower-dimensional vector v_{G_t} while preserving the properties of the original object with high probability.

Definition 3.4. **Cost vector** c_t is a vector of size $|S_t^k|$ where each element in c_t holds the cost of a discriminative shingle S_t^k

Using a projection vector h_d and a cost vector c_t, our goal is to embed a graph G_t into a $d-$dimensional sketch v_{G_t}. Such sketching will preserve good approximations of the original properties of the graph like graph proximity. In other words, if two graphs share a common structure, they share a higher number of n-shingles, therefore, their sketches v_{G_t} are closely mapped. Intuitively, graph proximity measures how many nodes, edges, and paths are shared by two graphs [2]. The graph proximity can then be calculated using similarity measures like Jaccard similarity, Cosine similarity, Euclidean distance, etc., between the set of sketches v_{G_t}. Give two graphs $G_1 = (v_1, e_1)$ and $G_2 = (v_2, e_2)$, the graph proximity between G_1 and G_2 is larger if the number of shared discriminative shingles is larger. Note that using sketches of n-shingles (if $n > 2$) for measuring graph proximity is not just similarity measures like Jaccard, Cosine, or Euclidean similarity measures of nodes or edges in the two graphs, as it also takes the connections among the common nodes into account.

3.1.2 Problem Definition. Given the aforementioned setting, we now formulate the problem that we aim to tackle in this work as follows: *Given a graph stream $G_s = \{G_1, G_2, ..., G_t, ...\}$, our goal is to learn a sketching function f for each graph $G_t \in \mathbb{R}^{|v|^2}$ such that $f : G_t \rightarrow v_{G_t} \in \mathbb{S}^d$, $d << |v|^2$ while ensuring that the original properties of the graph are well-preserved in such a $d-$dimensional vector v_{G_t}.*

The framework of our proposed method, SnapSketch , consists of two key steps: (i) generating shingles using a biased random walk, (ii) generating a fixed-sized $d-$dimensional sketch by hashing discriminative shingles. We now discuss these two steps in detail.

3.2 Shingling using Biased Random Walk

Random walks have been used as a similarity measure for a variety of problems in graphs [2, 14, 31]. Random walk can also measure the similarity between graphs by measuring the distance between two nodes in the graphs, the number of shared paths, the number of shared neighbors between nodes in the graphs, etc. Similarly, like a $k-$gram in a text document to construct vector representation [7], a random walk path can be decomposed into a set of $n-$shingles and a vector representation can be constructed using frequencies (or cost measures) of $n-$shingles. The similarity between the two graphs can then be defined on the similarity between their $n-$shingle. Hence, using random walks and shingling techniques, we can effectively capture the contexts in which graphs have high similarity.

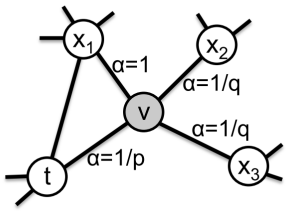

Figure 1: Pictorial representation of node2vec random walk [14]. The walk is at v (previously at t) and is evaluating its next step out of node v. Search biases α for each edges are shown as a label.

For each node v_i in graph G_t, the random walk of length l starting at node v_i is simulated in such a way that each i^{th} node in the walk path (w_i) is generated by the following distribution [14]:

$$P(w_i = x | w_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in e \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the non-normalized transition probability between node v and node x, and Z is the normalizing constant. The non-normalized transition probability can be calculated using edge weights. The normalizing factor Z, in this case, can be the total weight of all the edges between node v and x. We choose to do a short random walk starting from each node instead of a single long walk throughout the whole graph because if there is a disconnected component in the graph, the single longer walk might not traverse the disconnected part of the graph.

Neighborhood sampling using a breadth-first search (BFS) provides the structural information (like if the node is a bridge or a hub) about the local neighborhood. On the other hand, a depth-first search (DFS) can explore the global view of the graph and give information beyond the 1-hop neighborhood. Using node2vec [14], with a random walk we can smoothly interpolate between BFS and DFS and gather information about both local and global information of the node. The random walk can interpolate between BFS and DFS using parameters p and q. As shown in Figure 1, if a random walk just traversed the edge (t, v) and is currently at node v, the next step to the node x leading from the current node v on the walk can be decided using transitional probability $\pi_{vx} = \alpha_{pq}(t, x).w_{vx}$, where w_{vx} is the weight of the edge (v, x) and

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

where d_{tx} is the shortest distance between nodes t and x. The parameter p controls the likelihood of revisiting an older node in the walk while parameter q allows differentiating between "inward" and "outward" nodes. The high value of q provides a walk with

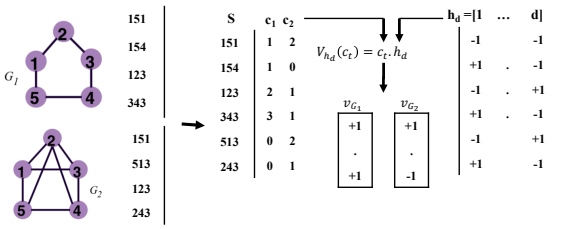

Figure 2: Hashing using shingle-frequency vector.

a local view (equivalent to BFS behavior) while a low value of q provides a walk away from the current node (equivalent to DFS behavior). Thus, we can get structural information efficiently (in both time and space) from a graph using a biased random walk [14]. Finally, we generate S_t, a set of n-shingles from the random walk path p_{G_t}.

3.3 Hashing Discriminative Shingle

Sketching techniques have been used as key features for representing graphs [3, 11, 26]. A graph G_t can be sketched using a frequency of n-shingles present in the graph. Let us take an example as shown in Figure 2 where S is the global shingle universe that holds all unique shingles in G_1 and G_2, and c_1 and c_2 are the shingle-frequency vector of size $|S|$ that holds the frequency of each n-shingle present in a graph. The similarity of graph G_1 and G_2 could then be measured using similarity measures between c_1 and c_2. However, in a real-time streaming scenario, whenever a newer graph G_{t+1} arrives in the stream, if the new *n-shingle* appears in G_{t+1}, then the number of dimensions for a sketch will increase (as the size of shingle universe $|S|$ increases). The sketch vector will increase to the exponential number of dimensions and is infeasible to compute or store in the long run. Hence, we propose a simplified hashing technique that composes a fixed-size graph sketch by hashing the top k discriminative *n-shingles* (instead of all shingles present in the graph).

The n-shingles obtained from a random walk path p_{G_t} can represent structural information of the graph. By effectively selecting the k most frequent n-shingles called discriminative shingles S_t^k, we can summarize or illuminate a major part of the graph. If k is low, only a part of the graph might be represented (only a few key neighborhoods will be summarized), while if k is sufficient enough most of the graph can be represented. The similarity between two sets of discriminative shingles can then be approximated using hashing such that:

- Hashing converts set of $k-$discriminative shingles into a small sketch (or signature) using a hashing function f, i.e., $f : S_t^k \rightarrow v_{G_t}$
- Sketch v_{G_t} is small enough to fit in the memory.
- If *similarity*$(S_t^k, S_{t'}^k)$ is high then Probability $P(v_{G_t} == v_{G_{t'}})$ is high.
- If *similarity*$(S_t^k, S_{t'}^k)$ is low then Probability $P(v_{G_t} == v_{G_{t'}})$ is low.

3.3.1 Simplified Hashing. For a graph G_t at time t, the top k discriminative shingles S_t^k are first instantiated with a $d-$dimensional

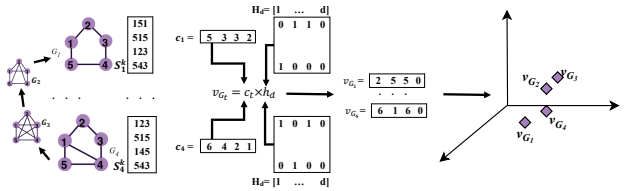

Figure 3: An illustration of the SnapSketch framework. The input is a collection of graphs (left). Using a fixed-length random walk, $k-$discriminative shingle S_t^k are generated. Simplified hash is used to generate the $d-$dimensional sketch vector v_{G_t} by multiplying cost vector c_t with projection vector h_d (mid). The similar graph are mapped closer using sketch vector (right).

projection vector $h_d = \{1, 0\}$ chosen uniformly at random with probability r such that:

$$\forall k \in S_t^k, h_{[1,\ldots,d]} = \begin{cases} 1 & \text{randomly chosen with probability r} \\ 0 & \text{randomly chosen with probability 1-r} \end{cases}$$

Using a shingle projection vector h_d, a graph G_t is embedded into a $d-$dimensional sketch vector v_{G_t} such that:

$$v_{G_t} = c_t \times h_d$$

where h_d is the $k \times d$ dimensional projection vector of S_t^k and c_t is the *cost vector* that holds the cost of $k-$discriminative shingles S_t^k in graph G_t such that:

$$c_t = [c_{s_1}, ..., c_{s_k}], \quad c_{s_i} = w_{s_i} \times r_{s_i}$$

where w_{s_i} is the sum of edge weights in the shingle S_i and r_{s_i} is the frequency of S_i in G_t. The shingle itself represents the neighborhood information while the cost will provide the structural information of the graph as we take the weight of the edges into consideration. Hence, a graph $G_t \in R^{|v|^2}$ can be represented by a fixed-size sketch vector $v_{G_t} \in S^d$ such that $d << |v|^2$. The visual illustration of the sketching technique is shown in Fig. 3. The pseudo-code of the proposed algorithm is given in Algorithm 1. The hash functions randomly project each discriminative shingle into a $d-$dimensional projection vector h_d (line-12). The sketch vector v_{G_t} for a graph G_t is obtained by multiplying the cost vector c_t with projection vector h_d (line 19). Each projection dimension can be considered as a snapshot that highlights and represents a key neighborhood of the graph (represented by shingles). Provided that we have enough snapshots, almost the entirety of the graph could be brought to light. The sketch vector v_{G_t} obtained using the above methodology can be used as an input to any standard machine learning algorithm for anomaly detection.

3.4 Complexity Analysis

In each time step t, two key steps are involved: generating a walk path using node2vec walk, and generating a sketch vector using shingles. First, for a node2vec walk, we perform a fixed-length walk from each node. Therefore, for a graph G_t with v nodes, the computation complexity would be $O(vl)$ where l is the length of

Algorithm 1: SnapSketch Algorithm

Input: Graph Stream $G_s = \{G_1, G_2, ..., G_t, ...\}$
Parameters: d, k, l, n
Output: *Anom_score*

Function `Main`(G_s, d, p, k, l, n):
  while *not end of stream* **do**
    $p_{G_t} \leftarrow$ `node2vecWalk`(G_t, l)
    $S_t \leftarrow \left[p_{G_t}[i : i+n] \text{ for } i \text{ in range } (\text{len}(p_{G_t}) - (n-1))\right]$
    $S_t^k \leftarrow S_t$`.sort(reverse=True)[:` k`]` `//get` $k-$`discriminative shingles`
    $h_d \leftarrow$`Hashing`(S_t^k, d, $r = 0.2$)
    $v_{G_t} \cup$ `Sketching`(S_t^k, h_d)
    Anom_score $\leftarrow$ RRCF(v_{G_t})
  end
Function `Hashing`(S_k, d, r):
  for $S_i = S_k[1, ..., k]$ **do**
    $h_d \cup$ `random([0,1], d, p=[1-r, r])`
  end
return h_d
Function `Sketching`(S_k, h_d):
  for $S_i = S_k[1, ..., k]$ **do**
    $c_t \cup w_{s_i} \times r_{s_i}$
  end
  $v_{G_t} = c_t \times h_d$
return v_{G_t}

the random walk. Second, to generate a sketch vector, $k-$ discriminative shingles are projected into a $d-$dimensional vector h_d that takes $O(kd)$ time, and generating a sketch vector v_{G_t} from the projection vector h_d and cost vector c_t takes $O(k^2 d)$. Therefore, the overall sketching time for a single graph with v nodes is $O(vl + k^2 d)$. Similarly, if there are n graphs in a graph stream G_s, it consumes $O(nd)$ space to store the sketch vector v_{G_t}.

4 ANOMALY DETECTION

Let us consider $G_t = (v, e, f)$ as a generic weighted graph at time t such that $\forall e$, a function f assigns a weight w to e, i.e, $f :\rightarrowtail e_{x,y} = w$ where x is the source node and y is the destination node. The neighborhood/region in the graph can be represented by the set of *n-shingles* generated from a random walk path. The localized regions/neighborhood of certain activities are called hotspots. The graph G_t is said to be anomalous if G_t exhibits the following behavior:

(1) If there is a sudden (dis)appearance of edge(s) with high weight. In other words, G_t is anomalous if $\exists e \in (G_t, G_{t-1}) \ni w_t >> w_{t-1} \vee w_{t-1} >> w_t$ where w_t is the weight of e in G_t and w_{t-1} is the weight of e in G_{t-1}.
(2) The sudden increase (or decrease) in incoming/outgoing edges to/from a vertex. In other words, G_t is anomalous if $\exists v \in (G_t, G_{t-1}) \ni N_t(v) >> N_{t-1}(v) \vee N_{t-1}(v) >> N_t(v)$ where $N_t(v)$ is the number of neighbors of v in G_t and $N_{t-1}(v)$ is the number of neighbors of v in G_{t-1}.

To summarize, the graph G_t is said to be anomalous if there is a sudden change in the localized graph structures (or hotspots) compared to the graphs $\{G_{t-1}, G_{t-2}, ...\}$ from the past. The important aspect of this definition is that the change in a hotspot should be sudden rather than slowly evolving, because slowly evolving communities are not a part of the anomalous hotspot. Detecting such anomalies is useful in identifying cyber-attacks like a port scan, denial of service attack, etc., in computer networks, unusual road traffic patterns related to holidays, events, accidents, etc., or a spammer in a social network that acts fast to increase their page (or account) activities. Since our random walk traverses the entire graph (a short walk from each node) and is biased toward the denser edges, the discriminative shingle if chosen effectively can take a snapshot of the significant portion of the hotspot in the graph. Therefore, as our focus is on identifying anomalous hotspots in the graph (like DoS attacks) we believe that the set of the most frequent k shingles can effectively summarize the graph. To quantify the effectiveness of a graph representation for an anomaly detection task, the sketch vectors are tested using an unsupervised anomaly detection algorithm.

4.1 Anomaly Detection Algorithm

The sketch vector obtained using SnapSketch and comparison approaches is given as an input to an anomaly detection algorithm. Anomaly detection is performed by using the state-of-the-art Robust Random Cut Forests (RRCF) [15]. RRCF algorithm is an unsupervised ensemble method for detecting outliers in streaming data. We initialize RRCF using 100 trees and 256 samples. RRCF initializes the forest of 100 trees by using the sketch vector of the first 256 graphs. RRCF outputs the anomaly score of a graph as a collusive displacement, which measures the change in model complexity incurred by inserting or deleting a given graph. A score above the 90^{th} percentile is marked as an anomaly. For a detailed description of RRCF, the reader can refer to [15].

4.2 Baseline Approach

The comparison graph stream sketching technique methods used in this work are StreamSpot [26] and SpotLight [11].

StreamSpot [26] StreamSpot introduces a similarity function that compares two graphs based on the relative frequency of local substructures represented as short strings. StreamSpot represents each graph G using a shingle-frequency vector z_G. A k-shingle $s(v, k)$ is constructed by traversing edges, in their temporal order, in the $k-$hop neighborhood of node v (also called Ordered $k-$hop Breadth-First Traversal). It then uses locality-sensitive hashing (LSH)-based similarly function to generate constant-space sketch vector representation of a graph in a stream. The LSH-based similarity function ensures that graph structures represented as a sketch vector preserves graph similarity. For more details about this approach, refer to [26].

SpotLight [11] SpotLight is a randomized sketching-based approach that guarantees that an anomalous graph is mapped 'far' away from 'normal' instances in the sketch space with a high probability for an appropriate choice of parameters. SpotLight composes a sketch containing total edge weights of K specific directed query subgraphs chosen independently and uniformly at random, according to node sampling probabilities, p for sources, and q for

Table 1: Network traffic dataset for anomaly detection.

Dataset	# of Graph	# of Anomalies	Edges	Time Unit
Smart Homes IoT	9,678	1,007	29,959,737	1 min
DARPA 1998	3,497	361	3,904,797	10 min

destinations. This leads to a (K, p, q)−SpotLight graph sketching. For more details about this approach, refer to [11].

4.3 Dataset

In this section, we evaluate the proposed methods using two publicly available time-evolving graph datasets that contain known anomalies. The datasets are related to network traffic and the anomalies are in the form of Denial of Service (DoS) attacks. A summary of each of the data sets is shown in Table 1.

Smart Home IoT Traffic The smart home IoT traffic dataset used in this study is collected from The University of New South Wales Sydney (UNSW Sydney) smart home test-bed [17, 36]. A real-life smart home environment was created with 28 different IoT devices that include cameras, switches and triggers, hubs, air quality sensors, electronics, healthcare devices, and light bulbs. For our experimentation, one week of data spanning across October 1-7, 2018 is used. Each DoS attack in the dataset lasts for 10 minutes and is launched with the maximum limit of 1, 10 or 100 packets per second. Based on this ground truth, if an individual graph contains at least 100 edges belonging to a DoS attack, the graph was labeled anomalous. It should be noted that the anomaly detection algorithm used is an unsupervised approach and the labeled dataset is only used for performance evaluation. For more detail on the process of data collection and data, the reader can refer to [36].

1998 DARPA Intrusion Detection Dataset [25] The DARPA 1998 datasets contain labeled data generated by simulating network traffic for a medium-size U.S. Air Force base. The dataset contains more than 300 instances of 38 different automated attacks in seven weeks of training data and two weeks of test data. The dataset contains denial of service (DoS) attacks designed to disrupt a host or network service. Some DoS attacks (e.g., smurf) excessively load a legitimate network service, others (e.g., teardrop, Ping of Death) create malformed packets that are incorrectly handled by the victim machine. For our experimentation, seven weeks of training data are used which contains 3,904,797 network traffic communications in the form of a TCP dump.

4.4 Data Preprocessing

A python-based script is created to parse the data. Network traffic datasets contain network traffic communication which can be mapped into a graph. A node in a graph represents network devices like DNS, web server, workstation, internet, etc., while an edge represents the unique communication between the devices. Each individual graph represents the traffic flow corresponding to the fixed time duration. One graph corresponds to 1 minute of traffic in Smart Home traffic and 10 minutes of traffic in the DARPA 1998 dataset. The duration is decided by evaluating the density of the traffic flow. If there are multiple communications between the same set of source and destination devices, we sum them and assign it as an edge weight. The edge weight will be used to calculate the transition probability of the next node in the random walk path. The higher the edge weight the higher the probability to go to the next node in a random walk. Similarly, the edge weight will be used to calculate the cost of the shingle in the graph. This is particularly important in identifying the denser subgraph where the attack device will be sending a high amount of traffic to the victim machine and it can be reflected as an edge weight. Using the aforementioned setting, a stream of 9,678 graphs for Smart Home IoT Traffic and 3,497 graphs for DARPA 1998 are obtained.

4.5 Experimental Setup

To capture the feature from the anomalous hotspot region (like the attack-victim device subgroup) of the graph, we interpolate our random walk as a Breadth First Search (BFS) by setting q higher than 1. Interpolating the random walk as a BFS would provide the local neighborhood information and the top k shingles generated from such a walk will constitute of a snapshot of the anomalous (denser) hotspot. The sketch vector is given as input to a state-of-the-art stream anomaly detection algorithm called Robust Random Cut Forests (RRCF) [15]. By default, SpotLight uses $K = 50$ sketch dimensions and $p = q = 0.2$ source/destination sampling probabilities. Similarly, StreamSpot uses the default chunk size = 24 for generating graph sketches. For SnapSketch , the length of random walk l is set to 50 and the size of shingle n is set to 3. After empirical evaluation, we decided to use $d = 64$, $k = 128$ for the DARPA dataset and $d = 32$, $k = 256$ for the Smart Home IoT dataset.

4.6 Results and Discussion

We now summarize the discoveries made by SnapSketch and the baseline approach.

4.6.1 Evaluation Metrics: The performance of the proposed approach and the baseline approaches is evaluated based on its effectiveness in identifying the DoS attack traffic. The ground truth of each dataset indicates that most of the DoS attacks are large (> 100 edges). Therefore, if a graph has a higher number of anomalous edges then they are more anomalous. It is particularly important in DoS attack scenarios where identifying the most anomalous graph means being more effective in detecting severe DoS attacks. For each graph in the stream, the number of anomalous edges is computed. Sorting these in descending order, the top m most anomalous graphs are identified. Each method is then evaluated on how well they can identify the top m anomalous graphs. If there are a total N anomalous graphs (graphs with DoS attack traffic), for every m we compute $precision@m = TP(m)/m$ and $recall@m = TP(m)/N$.

4.6.2 Results. Anomaly detection results on all three datasets are shown in Table 2. Precision and recall are reported for the top 100, 200, and 300 anomalous graphs. There are a total of 1,007 ground truth anomalies in the Smart Home IoT dataset. Ground truth values indicate the ideal score of precision and recall for each m. As shown in Table 2, our approach has better precision and recall compared against the baseline approaches for each value of m in the Smart Home IoT and DARPA datasets. Figure 4 plots the ground truth (blue plot) and the anomaly score (red plot) of all methods in the smart home IoT traffic from $t = 300$ to $t = 1200$.

Table 2: Performance comparison between SnapSketch , SpotLight, and StreamSpot on top−m anomalous graphs.

Algorithm	Precision (top−m)			Recall (top−m)		
	100	**200**	**300**	**100**	**200**	**300**
Smart Home IOT Dataset						
Ground Truth	*1.0*	*1.0*	*1.0*	*.099*	*.198*	*.298*
SnapSketch	**.94**	**.86**	**.80**	**.093**	**.170**	**.239**
SpotLight	.77	.73	.63	.076	.145	.190
StreamSpot	.69	.57	.54	.068	.114	.161
DARPA Dataset						
Ground Truth	*1.0*	*1.0*	*1.0*	*.277*	*.554*	*.831*
SnapSketch	**.83**	**.52**	**.34**	**.229**	**.288**	**.288**
SpotLight	.80	.51	.34	.221	.282	.282
StreamSpot	.49	.29	.20	.135	.160	.163

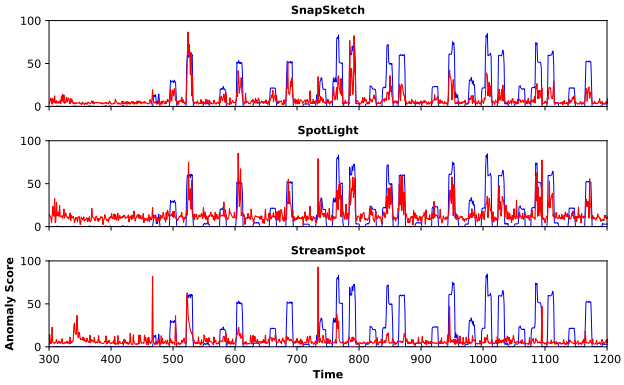

Figure 4: Anomaly score reported on smart home IoT traffic. Blue plot indicates the ground truth anomalies. Spike in red plots indicates the reported anomaly score.

The higher the spike on the blue plot the more anomalous is the graph. As shown, SnapSketch was able to identify most of the DoS attack spikes with better precision (red plot). Spotlight has fewer false alarms (on the left side of the plot) while the StreamSpot approach is missing several instances of DoS attacks. Similarly, SnapSketch consistently performs better than both SpotLight and StreamSpot for each value of m in the DARPA dataset. Figure 5 plots the ground truth (blue plot) and the anomaly score (red plot) reported by RRCF during the period $t = 1200$ to $t = 2600$ of the DARPA graph stream. Our approach was able to identify most of the attacks and had fewer false alarms. Like with the smart home IoT traffic, SpotLight has several false alarms and the StreamSpot approach misses several instances of attacks making SnapSketch a better approach in identifying intrusions in the DARPA dataset.

4.6.3 Discussion: Like SnapSketch , StreamSpot also involves hashing n−shingles. However, the shingles are generated from the walk path of an ordered k-hop breadth-first traversal rather than a biased-random walk. The performance of sketches given by StreamSpot on anomaly detection is persistently low on both datasets because

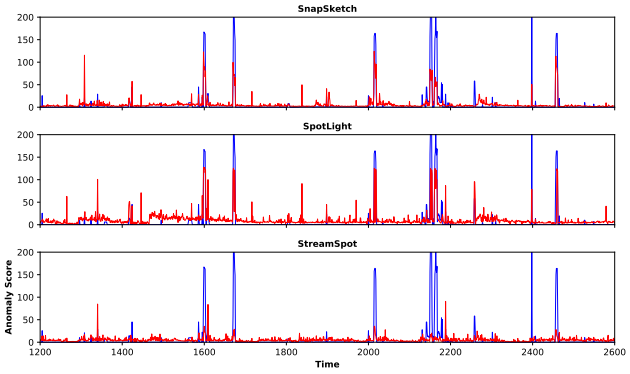

Figure 5: Anomaly score reported on DARPA dataset.

it was designed to detect anomalies in the form of host-level advanced persistent threats (APT) in security where system logs are used to construct an *information flow graph*. The anomalies in such scenarios are the deviation in system behavior rather than sudden (or significant) changes in the graph topology (or hotspot regions). Hence, it was unable to capture a DoS attack where the anomalies constitute a sudden change in the hotspot (burst of traffic in attack-victim subgraph regions). SpotLight is specifically designed to detect anomalies that constitute the sudden appearance or disappearance of dense subgraphs (like found in a DoS attack), and thus performs better than StreamSpot. SpotLight uses a K query subgraph for each graph by sampling source and destination nodes with probability p and q and sketches the graph into a K dimensional sketch vector. Each sketch dimension is called a "spotlight", where the anomalies would be brought to light by at least one of these spotlights. However, SnapSketch is able to outperform SpotLight. In SnapSketch , the random walk to generate shingles (also called the "snapshot" of the graph) is guided toward the denser edges (i.e., edges with higher edge weight have higher transitional probability) and are rendered like a breadth first search to capture enough of the local neighborhood. Using this strategy, sketches of SnapSketch can capture the sudden changes in the hotspot region. In summary, these results demonstrate the advantages of SnapSketch in identifying the DoS attack in network traffic.

5 CONCLUSION AND FUTURE WORK

In this work, we presented a graph representation technique called SnapSketch that can effectively represent graphs into a feature vector over time with efficient memory use. It first uses a biased random walk to generate shingles. The top discriminative shingles are then hashed into a d−dimensional projection vector h_d. Using the cost of the shingles and the projection vector, a graph is represented as a low dimensional sketch vector that is memory efficient and preserves the original properties of the graph. The learned sketches are then used for anomaly detection. Comprehensive experimentation of anomaly detection problems on well-known network traffic datasets demonstrates the meaningful and effective representations learned by our method. SnapSketch was able to outperform both baseline anomaly detection approaches while employed to detect network anomalies on streaming graphs.

SnapSketch performed better in identifying the severe type of DoS attacks, while the performance degraded when the attack strength is lessened (see Table 2). In the future, we would like to investigate a way to improve the performance when the attack strength is low. The biased random walk we used for traversing the graph provides the neighborhood information to calculate local and global neighborhoods, but the structure of the graph (number of neighbors, degree, etc.) is omitted. We would like to investigate an approach to integrate this structural information in our sketch vector. The learned graph representation was used for anomaly detection applications. In the future, we would like to test for other applications like graph classification, community detection, etc.

REFERENCES

[1] Bijaya Adhikari, Yao Zhang, Aditya Bharadwaj, and B Aditya Prakash. 2017. Condensing temporal networks using propagation. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 417–425.

[2] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. 2017. Distributed representations of subgraphs. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 111–117.

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. ACM, 5–14.

[4] Rami Al-Rfou, Bryan Perozzi, and Dustin Zelle. 2019. Ddgk: Learning graph representations for deep divergence graph kernels. In *The World Wide Web Conference*. 37–48.

[5] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*. 585–591.

[6] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.

[7] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[8] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 891–900.

[9] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[10] Giovanni Da San Martino, Nicolo Navarin, and Alessandro Sperduti. 2012. A memory efficient graph kernel. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–7.

[11] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. 2018. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1378–1386.

[12] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*. Springer, 129–143.

[13] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[15] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. 2016. Robust random cut forest based anomaly detection on streams. In *International conference on machine learning*. 2712–2721.

[16] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[17] Ayyoob Hamza, Hassan Habibi Gharakheili, Theophilus A Benson, and Vijay Sivaraman. 2019. Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity. In *Proceedings of the 2019 ACM Symposium on SDN Research*. ACM, 36–48.

[18] Zaïd Harchaoui and Francis Bach. 2007. Image classification with segmentation graph kernels. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

[19] Mohamed S Hassan, Bruno Ribeiro, and Walid G Aref. 2018. SBG-sketch: a self-balanced sketch for labeled-graph stream summarization. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. ACM, 3.

[20] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. 2004. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 158–167.

[21] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. 2003. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 321–328.

[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[23] Risi Imre Kondor and John Lafferty. 2002. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, Vol. 2002. 315–322.

[24] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. 2019. A Survey on Graph Kernels. *arXiv preprint arXiv:1903.11835* (2019).

[25] Richard Lippmann, Robert K Cunningham, David J Fried, Isaac Graf, Kris R Kendall, Seth E Webster, and Marc A Zissman. 1999. Results of the DARPA 1998 Offline Intrusion Detection Evaluation.. In *Recent advances in intrusion detection*, Vol. 99. 829–835.

[26] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. 2016. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1035–1044.

[27] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005* (2017).

[28] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[29] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1105–1114.

[30] Ramesh Paudel, Timothy Muncy, and William Eberle. 2019. Detecting DoS Attack in Smart Home IoT Devices Using a Graph-Based Approach. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 5249–5258.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[32] Jan Ramon and Thomas Gärtner. 2003. Expressivity versus efficiency of graph kernels. In *Proceedings of the first international workshop on mining graphs, trees and sequences*. 65–74.

[33] Nasrullah Sheikh, Zekarias Kefato, and Alberto Montresor. 2019. gat2vec: representation learning for attributed graphs. *Computing* 101, 3 (2019), 187–209.

[34] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, Sep (2011), 2539–2561.

[35] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*. 488–495.

[36] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* (2018).

[37] Alexander J Smola and Risi Kondor. 2003. Kernels and regularization on graphs. In *Learning theory and kernel machines*. Springer, 144–158.

[38] Nan Tang, Qing Chen, and Prasenjit Mitra. 2016. Graph stream summarization: From big bang to big crunch. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1481–1496.

[39] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *Journal of Machine Learning Research* 11, Apr (2010), 1201–1242.

[40] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[41] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1365–1374.

[42] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2672–2681.

[43] Peixiang Zhao, Charu C Aggarwal, and Min Wang. 2011. gSketch: on query estimation in graph streams. *Proceedings of the VLDB Endowment* 5, 3 (2011), 193–204.

[44] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. 2019. Addgraph: anomaly detection in dynamic graph using attention-based temporal GCN. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 4419–4425.