8

Anomaly Detection in Networks

LEARNING OBJECTIVES

After completing the chapter, the readers are expected to

- Learn various types of anomalies at different levels—nodes, edges, subgraphs.
- Compare and contrast various anomaly detection methods.
- Define new types of anomalies in a given network.
- Design algorithms for efficient and cost-effective anomaly detection.

Consider that you are an active participant in the political debates and discussions on different social media platforms—Twitter, Reddit, and Quora. You are a strong support of democrat and always argue against republic. In general, an online discussion takes place with participants posing views of both the political parties. Some of the participants take a neutral role and primarily act as listeners till the end. However, you have been noticing in the few discussions that a few participants join the discussions at the middle, pose soft and neutral opinion initially, and gradually become very aggressive with the intention to dominate the discussion with their political views. You have also noticed the same group of participants surprisingly supporting democrats in one discussion, and republicans in an other discussion, both very aggressively—they do not follow any firm ideology as such. Still whenever they comment, they express a firm opinion. Being a frequent participant in such discussions, you may find such behaviour very different from the usual one. In social network analysis, such participants are known as "anomaly".

In the various domains, an interesting problem statement that is widely studied is spotting "anomalous" behaviour that stands out from the general trend. This is referred to as "anomaly detection", constituting an essential component of data mining. Formally, an anomaly can be defined as a pattern or behaviour that deviates from the expected trend. In other words, it does not conform to the standard observations. There are numerous applications in which anomaly detection plays a significant role. These include, but are not limited to, credit card fraud detection, false

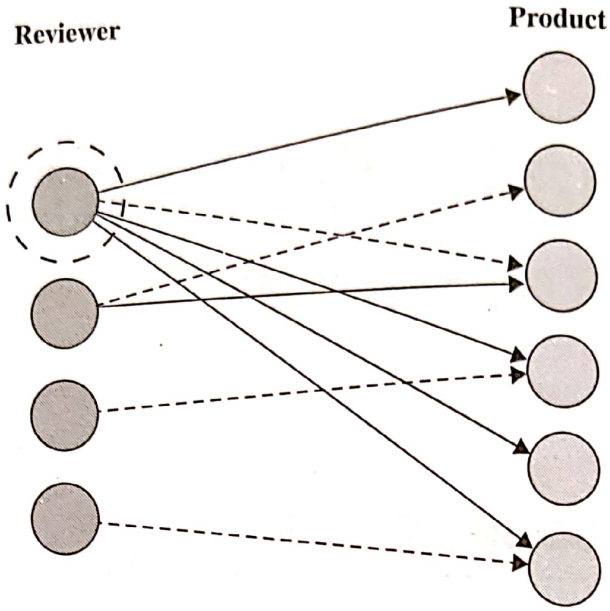


FIGURE 8.1. Anomaly in a reviewer-product opinion network. Solid lines edges indicate negative reviews, and dotted line edges indicate positive reviews.

advertisement, network intrusion, etc. In this chapter, we take a look at network-based anomaly detection wherein the dataset is available in the form of a network made up of nodes and edges. Network-based anomaly detection itself has specific applications, for instance, finding fake reviews in an opinion network. Figure 8.1 presents a toy example of a reviewer-product opinion network. It can be seen that the encircled reviewer gives a negative review to all the products except one, suggesting that she might be recruited by the manufacturer of the positive review product for giving bogus reviews to its competitors. Studies around network-based anomaly detection include finding unusual behaviour in the financial trading networks (Li et al. 2010), spotting web network spammers (web pages gaining false importance in the form of high PageRank scores) (Gyöngyi et al. 2004), or in social networks (for instance, adding a lot of friends on Facebook to win a social media popularity contest). Anomaly detection, apart from revealing malicious behaviour, is also helpful in pre-processing in the form of data cleansing by removing specific points that may not fit the underlying distribution.

In this chapter, we begin with a general discussion on how anomalies in a network-based setting differ from the outliers. This is followed by a brief overview of challenges involving the development of network-based anomaly detection algorithms. We then dig deeper into various state-of-the-art methodologies that have been proposed in the literature for different types of networks: (a) static networks (plain and attributed), and (b) dynamic networks, based on different aspects such that network structure, node clustering, node/edge attributes. This chapter is highly motivated by the comprehensive survey of Akoglu et al. (2015). We provide an overview of various types of network-based anomaly detection methods. Figure 8.2 gives a brief representation of the taxonomy.

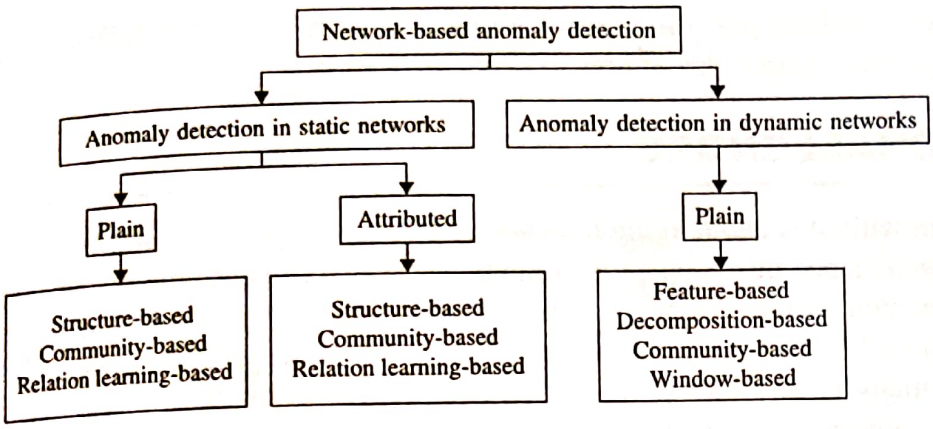


FIGURE 8.2. A taxonomy of network-based anomaly detection methods.

8.1 OUTLIERS VERSUS NETWORK-BASED ANOMALIES

Outlier detection is a famous field of research in the domain of data mining. However, it is significantly different from network-based anomaly detection. In the network-based anomaly detection, the dataset is mapped into a network, and the anomalies are detected based on this network dataset generated.

In regular outlier detection, the dataset is generally mapped into a feature space, i.e., converted into a vectorised representation, and then processed further to detect outliers. On the contrary, in a network-based anomaly detection, there is no such concept of mapping the dataset into a feature space or generating vectorised representations for the data points.

One of the key characteristics of network-based anomaly detection is that the network captures *inter-dependencies* among nodes in the network, whereas, in regular outlier detection, the points are considered independently in a multi-dimensional space.

If we look at a classification problem, e.g., a support vector machine model, the method involves mapping the data points into a feature space and then classifying them. In such a method, we automatically assume the data points to be independent (drawn from an independent and identical distribution). As a result, all the inter-dependencies of the data points are lost or ignored.

Researchers argue that network-based anomaly detection is better-suited for finding anomalous data points. For instance, in a reviewer-product network, considering a review independently might not give us any reliable answer as to whether the review is genuine or fake, i.e., if the product is how the review claims it to be. However, if we look at all the reviews together, we would be able to spot a general trend—if the product is good, most reviews will claim so and vice-versa. The reviews that significantly deviate from the general trend would then be flagged as anomalous. The same problem can also be better solved by looking at the other products that the same reviewer has rated. In the fraudulent rating of products, people rate all the products of a specific type negatively other than the one they are paid to promote.

1.outlier detection no use ?
2.how to do the detecction in both of these approaches ?
3.

approach

what is the disadvantage of svm?

Therefore, looking into the reviews posted by the reviewer for different products could give us a better idea of how genuine her review is.

8.2 CHALLENGES

To begin with, it is *challenging to generate a concrete definition of what constitutes an anomaly* since this concept is domain and task-specific. Some scenarios involving anomalous behaviour of network entities include: (a) a node having a rare combination of features in an attributed network, (b) isolated nodes located far away from a majority of nodes in the network, or (c) nodes that are 'surprising' and so may not adhere to the underlying distribution of our network model. Along with this, real-world networks available today are huge, e.g., social networks such as Twitter and Facebook have billions of users. Adding to this, these real-world *networks grow dynamically* with new information arriving at a very high rate. Network entities, namely, nodes and edges, have a lot of metadata information associated with them in the form of complex feature vectors. For anomaly detection, usually, there is a *lack of labelled datasets* wherein distinct labels are available for data points that are anomalous and non-anomalous. Manual labelling is not a feasible option in this case since a lot of metadata in the form of historical behaviours, or domain knowledge of a data point may be required to judge whether it can be classified as an anomaly or not. Human interpretation is very subjective. Anomalies are primarily considered as 'rare' occurrences, introducing a *class imbalance* in the underlying data. As mentioned earlier, networks today are changing and growing dynamically. Due to this, anomaly detection algorithms should be *capable of finding 'novel anomalies'* in different evolving settings of the same dataset. In addition to this, the detection algorithms should be *robust enough* to understand different types of anomalies that could fit into a network. Moreover, after extracting anomalies, it is equally *important to explain why they are flagged as anomalous*, making it easy for outsiders to get a clear understanding. Network-based anomaly detection algorithms should take into consideration all the aforementioned challenges.

8.3 ANOMALY DETECTION IN STATIC NETWORKS

Let us begin our discussion with network-based anomaly detection methods for static networks, i.e., given a non-evolving snapshot of a network, we are interested in finding certain entities in the network that pose as an anomaly or, in other words, show a behaviour that is different from the usual patterns exhibited by the network.

As shown in Figure 8.2, static networks can be of the following two types: (a) plain networks, and (b) attributed networks.

Plain networks simply consist of nodes and edge connections. On the other hand, attributed networks have additional information/features associated with the nodes and/or edges. For instance, Figure 8.3 provides a toy example for an attributed network such that nodes prefer a media streaming service (Netflix or Amazon Prime Video), and edges have weights representing the strength of a connection.

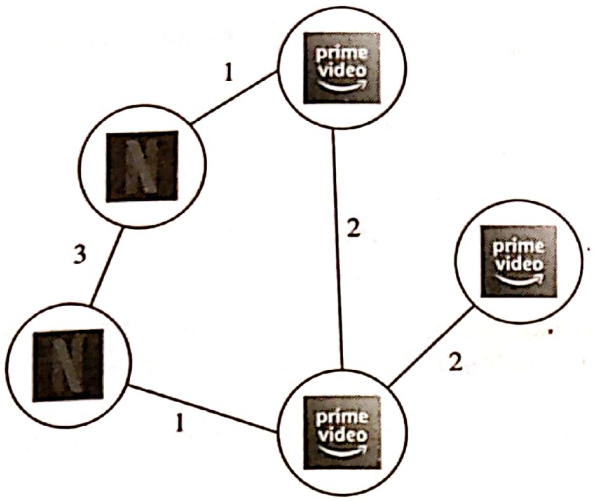


FIGURE 8.3. A toy example of an attributed network.

8.3.1 Plain Networks

Let us begin with the anomaly detection algorithms for plain static networks. These algorithms mainly take into account the network structure to detect anomalies. We shall primarily explore the following two types of patterns to spot anomalies: structure-based patterns and community-based patterns.

1. Structure-based Approaches

The approaches can be classified into the following two main classes: (a) feature-based and (b) proximity-based.

(a) **Feature-based approaches:** These algorithms are centred towards utilising the network structure to extract certain network-centric features for anomaly detection. Table 8.1 lists some of the network-centric features that can be used to spot anomalies.

TABLE 8.1. A few network-centric features.

Network-based features				
Node level	**Dyadic**	**Egonet**	**Node group**	**Global**
Eigenvector Closeness Betweenness Local clustering coeff. Degree assortativity	Reciprocity Edge betweenness Common neighbours	No. of triangles Total weight Principal eigenvalue	Density Modularity Conductance	Global clustering coeff. Avg. node degree No. of connected components MST weight

Let us now discuss one of the state-of-the-art feature-based approaches to detect anomalies in a plain static network, called ODDBALL (Akoglu et al. 2010). ODDBALL extracts egonet-based features to focus on the subnetwork induced by the immediate neighbours of a node. Therefore, it looks for egonets that exhibit an anomalous behaviour, which in turn reduces to finding anomalous nodes in the network.

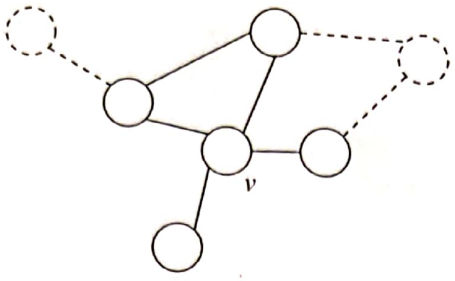


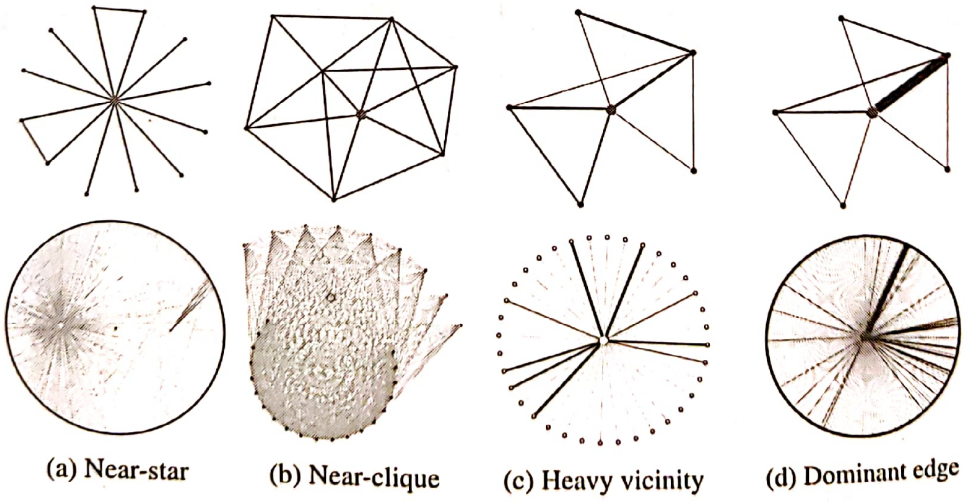
FIGURE 8.4. A toy example of an egonet. Given the ego node v, its egonet is represented by the solid nodes and edge connections between them.

Egonet

Given a node (referred to as *ego*), its egonet is defined through a subnetwork induced by its 1-hop neighbours (referred to as *alters*) or its immediate neighbours and the node itself. Figure 8.4 provides a toy example for an egonet.

ODDBALL majorly looks at four types of anomalies in a given network:

1. *Near-cliques and near-stars:* Nodes/egos whose neighbours are highly connected amongst each other (near-cliques: Figure 8.5(b)) or sparsely connected amongst each other (near-stars: Figure 8.5(a)) are considered to be "strange" or anomalous.
2. *Heavy vicinities:* In a *who-calls-whom* network, given that an individual u calls n distinct individuals, the total number of calls made by her should be approximately $O(n)$. An extreme number of phone calls (in this case total weight of edges, Figure 8.5(c)) turn out to be anomalous/suspicious, e.g., multiple redials because of a bad telephone service.
3. *Dominant heavy links:* In the same *who-calls-whom* network, if an individual u contacts one of its immediate neighbours, v constantly, i.e., *dominating* its connection to single node v, then it poses as a suspicious/anomalous activity (Figure 8.5(d)).



(a) Near-star (b) Near-clique (c) Heavy vicinity (d) Dominant edge

FIGURE 8.5. Anomalous substructures considered in ODDBALL. Reprinted with permission from Akoglu et al. (2010).

Given that ODDBALL tries to search for the above-stated anomalies using egonet-based features, let us now understand *what all features* ODDBALL considers to define the neighbourhood or egonet of a node to extract anomalous entities.

Among a whole set of network-centric features (some are listed in Table 8.1), it focuses on those that (a) are efficient in terms of computation and (b) provide suitable patterns describing "normal" behaviour of a neighbourhood structure.

The set of features chosen are as follows:

1. N_i: Number of neighbours or degree of the ego node i.
2. E_i: Number of edges in the egonet of node i.
3. W_i: Total weight of the egonet (edges in the egonet) for node i.
4. $\lambda_{w,i}$: The principal eigenvalue of the weighted adjacency matrix of the egonet of node i.

ODDBALL is based on the following three observations used to define nodes with "normal" neighbourhoods. Egonets of nodes that deviate from the defined "normal" behaviours are declared as anomalies.

Observation 8.1

The number of neighbours N_i and the number of edges E_i for an egonet G_i follow a power law such that,

$$E_i \propto N_i^{\alpha}; 1 \leq \alpha \leq 2 \tag{8.1}$$

Observation 8.2

The total weight W_i and the number of edges E_i for an egonet G_i follow a power law such that,

$$W_i \propto E_i^{\beta}; \beta \geq 1 \tag{8.2}$$

Observation 8.3

$\lambda_{w,i}$, the principal eigenvalue of the weighted adjacency matrix and the total weight W_i for an egonet G_i follow a power law such that,

$$\lambda_{w,i} \propto W_i^{\gamma}; 0.5 \leq \gamma \leq 1 \tag{8.3}$$

Later, Henderson et al. (2011) extended ODDBALL to propose another algorithm called **ReFeX** (Recursive Feature Extraction) by combining local node-based features and egonet-based features for network-based anomaly detection.

(b) **Proximity-based approaches:** These algorithms are based on quantifying the closeness of nodes in a network in spotting anomalies.

Here, we provide a brief overview of the techniques used in this category of approaches since a detailed description of these has already been covered in the previous chapters.

(i) **PageRank:** PageRank (Brin and Page 1998) is an algorithm that calculates the *importance* of a node based on the *importance* of its neighbouring nodes. If a node has an extremely high rank (importance), it could be a sign of an anomaly.

(ii) **Random walk with restart (RWR):** This is a random walk based algorithm in which, at each step, the random walker has a small probability of jumping back to the source node of the walk (thereby *restarting* the random walk) in addition to the traditional transitional probabilities based on PageRank. This helps in recognising a neighbourhood of nodes in proximity to the source node.

(iii) **Personalised PageRank/topic-sensitive PageRank:** This algorithm (Haveliwala, 2003) is similar to RWR. Instead of having a small probability of restart (going back to the source node), each node is associated with a chance of getting "teleported" to one of the nodes belonging to the given "teleportation set" S. In case of RWR, S only consists of the source node, i.e., $|S| = 1$.

(iv) **SimRank:** SimRank (Jeh and Widom 2002) is a measure of similarity that claims that two nodes in a network are similar if they are further "referenced" by similar nodes.

(v) **Jaccard proximity:** Jaccard index is the ratio of the number of common neighbours between two nodes to the number of nodes in the union of neighbour sets of the two nodes. It is a measure of closeness or proximity, denoting the likelihood of an edge existing between the two nodes.

Many such proximity indices such as Common Neighbours, Adamic Adar and Preferential Attachment that have already been studied in Chapter 6, fall in the category of proximity-based approaches for the network anomaly detection.

2. Community-based Approaches

The approaches are based on detecting communities (clusters or closely connected nodes) to spot anomalous nodes and/or edges with a large number of cross-community relations. Such anomalies are known as bridge nodes/edges that do not directly belong to any one community. Here, we shall mainly discuss two state-of-the-art approaches.

(a) **Anomaly detection in bipartite networks (Sun et al. 2005):** This method takes its motivation from the existence of a lot of real-world networks as "bipartite networks". For instance, a publication network can be modelled as a bipartite network with two partitions of nodes: (a) authors and (b) papers written by the authors.

Bipartite network. Recall that bipartite networks are those networks that have two sets of nodes with no internal connections among the set members. In other words, no intra-set link is present, all the links are inter-set links. Figure 8.6 provides a toy example for a bipartite network.

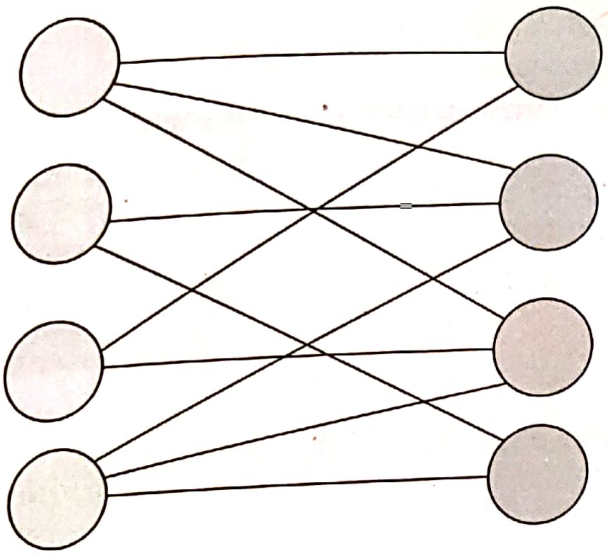


FIGURE 8.6. A toy example of a bipartite network.

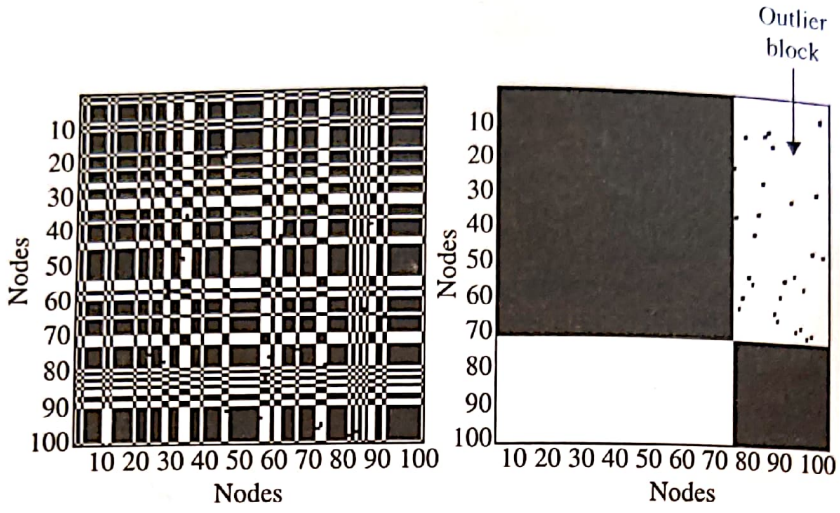
Sun et al. (2005) studied interesting properties shown by bridge nodes in the bipartite networks. For instance, in the publication network discussed earlier, we look for "unusual/anomalous" papers written by authors belonging to the different research communities. A couple of fundamental questions are as follows:

(i) How to detect the community (also referred to as the "neighbourhood") of a given node?

(ii) How to quantify a measure to assess the level at which a node should be considered as a bridge node?

To detect the neighbourhood of a given node, Sun et al. (2005) used the RWR-based Personalised PageRank (PPR). For a given node, the algorithm computes the PPR scores for all the nodes in the network such that nodes with the highest PPR scores form the neighbourhood of the given node.

The PPR scores obtained with respect to a given node also help in quantifying a measure to spot bridge nodes in the network. Given a node, pairwise PPR scores amongst all its neighbours are averaged to form a "normality" score for the node. Sun et al. (2005) claimed that the nodes with lower normality score constitute bridge nodes. The intuition behind this is that, for a node, that belongs to a single community, its neighbours will be visited many times during PPR. Since they belong to one community, the random walker will keep on visiting the same community nodes (highly overlapping neighbours) and thereby increasing the PPR scores for the neighbour nodes. On the other hand, for a bridge node, the neighbours are sparsely connected. So, once the random walker takes a step and goes into a particular community, it will get stuck there and its other neighbours (belonging to other communities) will not be visited very often. Therefore, not all the neighbours of a bridge node get enriched in their PPR scores, and sum to a value lesser than the non-bridge nodes. In other



(a) Outlier-test graph (b) Node groups and outliers

FIGURE 8.7. Representation of how AUTOPART works to spot anomalies in a network. Reprinted with permission from Chakrabarti (2004).

words, for bridge nodes, the neighbours have low-pairwise proximity due to a lot of cross-community connections.

(b) **AUTOPART (Chakrabarti 2004):** Similar to Sun et al. (2005), AUTOPART flags cross-community nodes and/or edges as anomalies. To get node clusters, the authors used the Minimum Description Length (MDL) (Rissanen, 1999) principle to rearrange the rows and columns of the network adjacency matrix-forming dense blocks/clusters of highly connected nodes. The low-density blocks are flagged as outlier/anomalous blocks, as shown in Figure 8.7.

Some other methods that lie under the domain of community-based anomaly detection for plain networks include Tong and Lin (2011), Ding et al. (2012), and Xu et al. (2007). Readers are highly encouraged to read these articles.

8.3.2 Attributed Networks

As described earlier with the help of Figure 8.3, attributed networks have an additional information in the form of node and/or edge attributes/features in addition to their basic network structure. The network anomaly detection algorithms under this domain take into account the basic network structure and meaningful insights extracted from node and/or edge attributes to detect unusual behaviour (or anomalies). Similar to plain networks discussed in Section 8.3.1, we shall primarily explore two types of patterns to spot anomalies: structure-based patterns and community-based patterns.

1. Structure-based Approaches

Under this category of approaches, algorithms aim to find infrequent substructures/subnetworks (essentially those that deviate from the "normal" behaviour) based on their structural connectivity and attribute information.

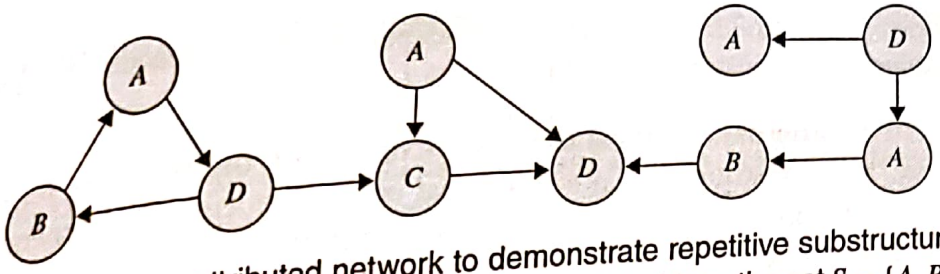


FIGURE 8.8. A toy attributed network to demonstrate repetitive substructures in a network. Each node in the network has a categorical label from the set $S = \{A, B, C, D\}$.

To begin our discussion, let us start with a pioneer method proposed by Noble and Cook (2003) for detecting anomalies in static networks having categorical attributes/labels. They essentially examined two different problem statements around finding network anomalies:

(a) To extract anomalous/unusual substructures in a network.
(b) To extract rare subnetworks from a set of subnetworks such that the nodes and/or edges contain categorical labels (may not be unique across all the entities).

In order to arrive at a solution for the aforementioned two problem statements, there is a need to define the usual "best" substructures/subnetworks to declare those as anomalies that do not follow the criterion.

Noble and Cook (2003) defined "best substructures" by gaining motivation from the core principle of Subdue (Cook and Holder 2000). Subdue algorithm primarily detects patterns or substructures that repeat in a network. For instance, in Figure 8.8, it can be seen that the substructure $A \rightarrow D$ repeats twice in the network.

The basic idea is that "best substructures" are defined as those that occur frequently in the network. Such substructures, thereby, help in *compressing* the network better.

Network Compression

Compressing a network structure refers to replacing substructures or subnetworks with a new node denoting the replaced substructure.

To evaluate the compression performance of a substructure, i.e., how well it is able to compress the network, preserving the network information quality, MDL (Rissanen, 1999) governed by Equation 8.4 is used. The best substructures can be obtained by minimising Equation 8.4.

$$F(S, G) = DL(G|S) + DL(S) \tag{8.4}$$

where S is a substructure for network G such that $DL(G|S)$ represents description length after compressing G using S, and $DL(S)$ represents description length of the substructure.

Keeping this in mind, let us now delve deeper into the proposed solutions for the two problem statements mentioned here:

(a) **Extract anomalous/unusual substructures in a network:** As seen earlier, the "best substructures" are those that occur frequently and generate lower values for Equation 8.4. Anomalous substructures can now be identified such as that produce relatively higher values for Equation 8.4. As a result, an inverse variant of the MDL measure would act as an appropriate quantity that can then be minimised to determine unusual substructures.

It is worth noting that infrequency may not be a sufficient measure for anomaly detection. For instance, one may consider the entire network as a substructure that is the least frequent (occurring only once) among all, but still should not be flagged as an anomaly by the system! To overcome this drawback of Equation 8.4, one can take a heuristic, given in Equation 8.5 that approximates as an inverse of Equation 8.4 (with an increase in $F(S, G)$, $F'(S, G)$ decreases), taking into account a balance between the size of the substructures and the quality of network compression that they provide.

$$F'(S, G) = \text{size}(S) \times \text{instances}(S, G) \tag{8.5}$$

where size(S) represents the number of vertices in substructure S, and instances(S, G) represents the number of times S appears in G.

As a result, substructures with low values for Equation 8.5 would be declared as anomalous. This metric will not flag the entire network as anomalous. For instance, taking the entire network of Figure 8.8 as S, $F'(S, G) = 10$, the highest value that can be obtained for any substructure, making it the least anomalous. On the other hand, a single node substructure with label C would be flagged as anomalous having $F'(S, G) = 1$.

(b) **Extract rare subnetworks from a set of subnetworks:** The main idea that governs the approach for solving this problem statement is that, subnetworks having a lot of commonly occurring substructures are less unusual or anomalous as compared to those with only a few common substructures. One can use `Subdue` in an iterative manner to find the best substructures using Equation 8.4, and then compress the subnetworks with them. Anomalous subnetworks would, thereby, experience less compression in comparison to the normal ones since they would contain less of these frequent or common "best substructures" obtained by `Subdue`.

The aforementioned methodology, proposed by Noble and Cook (2003), only deals with networks with categorical attributes. Nodes with unique numerical features in a network would appear to be equally anomalous under this setting. Real-life networks could be a mix of both the categorical and numerical attributes. To address this problem, Davis et al. (2011) proposed `Yagada`, an adaptation of `Subdue` for networks with numerical attributes. It discretises the numerical attributes such that the "normal" numerical attributes are all assigned the same categorical label, and the anomalous attributes get their "outlierness score".

2. Community-based Approaches

This type of algorithms aims to find "community outliers" or anomalous nodes that significantly differ from other community members based on their attribute values, e.g., a swimmer in a group of musicians consisting majorly of non-swimmers. There are two types of techniques—(a) identify outliers along with detecting communities and (b) detect communities in attributed networks first and then extract anomalies.

In order to motivate readers on how the community-based network anomaly detection algorithms function for attributed networks, we present an in-depth discussion of FocusCO (Focused Clustering and Outlier Detection) (Perozzi et al. 2014) that extracts anomalous nodes while detecting "focused" clusters in a user-interactive manner. Let us first understand what is meant by "focused" clusters.

Focused Clusters

In the case of attributed networks, users can themselves specify what all attributes should be considered for clustering a network, allowing them to drive the network clustering. To do so, users provide a set of "exemplar nodes" that they perceive to be similar. Attribute weights are then inferred from these exemplar nodes, and those with large weights are termed as "focus attributes" used for performing focused clustering.

Example 8.1

Consider a network, as shown in Figure 8.9, consisting of nodes with three attributes: (a) highest degree attained, (b) location of residence, and (c) the place of work. There are two focused clusters: (a) based on similar degree, and (b) based on similar place of work. On the basis of the user preferences, different focused clusters would be discovered. For instance, if a user is interested in extracting nodes with similar degree, only the group of nodes on the left will be discovered as a focus cluster. It should also be observed that the node with attribute as 'B.Tech.' would be flagged as a "focused outlier". Though it structurally belongs to the left cluster, it significantly differs from the other members on the basis of the focus attribute "the highest degree attained".

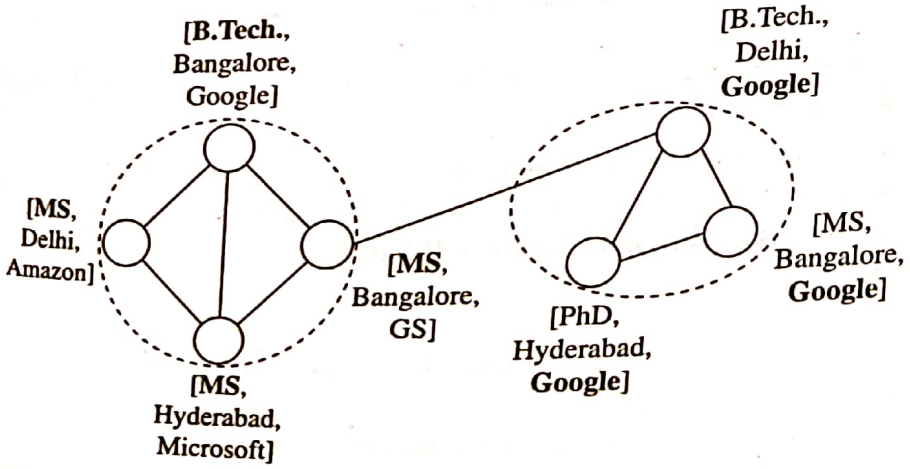


FIGURE 8.9. A toy example with two focused clusters—(a) the left one—based on attribute "highest degree attained", and (b) the right one—based on attribute "place of work".

Overview of FocusCO: Given an attributed network $G = (V, E, F)$ such that V represents the set of nodes, E the set of edges, and F the set of attributes associated with all nodes in G, FocusCO outputs only those focus clusters C of G that correspond to the interests of a user u. To do so, u provides C_{ex}, a set of exemplar nodes that are similar to those present in C (focus clusters preferred by u). Using G and C_{ex} as input, FocusCO performs the following:

1. **Infers importance/weights (β_u) of all the attributes in F based on C_{ex}:** This *extracts* specific *focus attributes* that define the similarity of nodes in C_{ex}. For instance, if C_{ex} contains four nodes present in the left cluster of Figure 8.9, the weight of attribute "the highest degree attained" would be maximum since it forms a basis of defining their similarity, making it a focus attribute. Refer Algorithm 8.1 to understand the internal working of how β_u is obtained.

Algorithm 8.1 InferAttributeWeights

Input: Network $G = (V, E, F)$, exemplar nodes C_{ex}
Output: Attribute weight vector β_u
1 Initialisation: Similar pairs $P_S = \phi$, Dissimilar pairs $P_D = \phi$;
2 **for** $u \in C_{ex}$, $v \in C_{ex}$ **do**
3 $\quad |P_S = P_S \cup (u, v)$
4 **end**
5 **while** $|P_D| \neq |F||P_S|$ **do**
6 $\quad$ | Sample $u \in V \backslash C_{ex}$
7 $\quad$ | Sample $v \in V \backslash C_{ex}$
8 $\quad |P_D = P_D \cup (u, v)$
9 **end**
10 Oversample P_S such that $|P_D| = |P_S|$
11 Get A by solving the objective function,
$$\min_A \sum_{(i,j)\in P_S} (f_i - f_j)^T A (f_i - f_j) - \gamma \log\left(\sum_{(i,j)\in P_D} \sqrt{(f_i - f_j)^T A (f_i - f_j)} \right)$$
12 return $\beta_u = \text{diag}(A)$

   (a) Similar node pairs (P_S: pairs $\in C_{ex}$) and dissimilar node pairs (P_D: pairs $\notin C_{ex}$) are obtained.
   (b) Diagonal matrix A corresponding to weights of all the attributes in F is learned using an objective function based on Mahalanobis distance (see line 11 of Algorithm 8.1) between the feature vectors of two nodes (f_i,f_j for node pair (i,j)),
$$\text{dist}(i,j) = (f_i - f_j)^T A (f_i - f_j) \tag{8.6}$$
   (c) β_u is then set to the diagonal entries of A.
2. **Extracts focus clusters C in G** using the attribute weights vector β_u obtained earlier. C consists of clusters that are: (a) densely connected internally and sparsely connected from the rest of G, and (b) have nodes that are similar with respect to the focus attributes. Initially, candidate core sets are identified with

the help of Algorithm 8.2, which are then further expanded until there is an improvement in the cluster quality.

Core expansion. To expand each core set C obtained from Algorithm 8.2, the non-member neighbours of nodes in the core set are considered. In order to evaluate whether adding a node n improves the quality of the cluster or not, weighted conductance (Leskovec et al. 2009) is used, given by Equation 8.7.

$$\phi^w(C,G) = \frac{\sum_{(i,j)\in E,\, i\in C,\, j\in V\setminus C} w(i,j)}{\sum_{i\in C}\sum_{j,(i,j)\in E} w(i,j)} \tag{8.7}$$

Using greedy optimisation at every iteration, node n that brings the largest drop in weighted conductance of cluster C is added to it. Following this, it is seen if removing certain nodes in C can improve its quality.

Algorithm 8.2 `FindCoreSets`

Input: Network $G = (V, E, F)$, attribute weights vector β_u
Output: Cores
1 Re-weigh edges in E using feature similarity of end-nodes
2 Induce a subnetwork g of G with edges having comparatively higher weights
3 **return** Connected components in g

3. **Detects focused outliers** O such that they structurally belong to focus clusters C in G, but differ significantly from other cluster members based on the focus attributes. To capture this, we consider nodes during core expansion that are structurally best (referred to as BSN), determined using unweighted conductance. As a result, nodes that are in BSN but not in C end up being detected as anomalies in set O.

8.3.3 Relational Learning

Algorithms of this category consist of network-based collective classification. The underlying principle is to exploit the complex relationships between the data points to assign them into appropriate classes, which are usually binary in nature—normal and anomalous. Intrinsically, it can be formulated as a classification problem. These methods are different from the methods based on proximity that deal with quantification of autocorrelation among nodes in attributed networks and are also more complex in nature, e.g., classify whether an online page is a spam page or not based on the keywords that appear on the page and then identify whether it is a benign page or not, similarly with the fraud detection problem. Combining all the research done in relational classification methods, generally, these approaches exploit the following inputs:

1. Unique attributes (features) of the nodes
2. Pre-labelled class of the node's neighbours
3. Unique attributes of the node's neighbours

Combining all the relational inference algorithms, they can be broadly listed as,

1. Loopy belief propagation
2. Gibbs sampling
3. ICA (iterative classification algorithm)
4. Weighted-vote relational network classifier

All of these algorithms are known to be fast and are approximate since the problem of exact inference is known to be NP-hard in the real-world networks and naturally, for any of the algorithms, its convergence is not guaranteed.

8.4 ANOMALY DETECTION IN DYNAMIC NETWORKS

So far, we have studied how anomalies can be detected in static networks. However, real-life systems are quite different. They constantly change over time. This also holds true for networks, wherein relationships between different nodes can change.

Networks that change over time are called *dynamic networks*. Usually, the set of nodes remains constant and only edges change; however, nodes can also change (get added or deleted) with time.

Most commonly, dynamic networks are created when an 'event' or 'change' occurs as stimuli and issues a change in the nodes or the edges. A dynamic network setting is denoted by a *sequence* of static networks, which maintain a temporal order. An example sequence is shown in Figure 8.10.

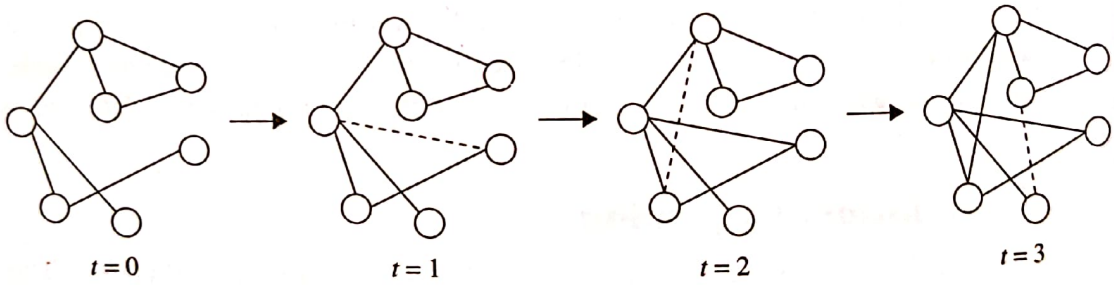


FIGURE 8.10. Dynamic network—edge set changes with time.

8.4.1 Preliminaries

In order to detect anomalies in dynamic networks, we consider the following. We are given a temporal sequence of plain or attributed networks.

1. We first identify the timestamps at which a sudden change or event occurs, which forces in changing the network.
2. We then extract the top k-nodes, edges or parts of the network that contribute most to that change or event across two subsequent timestamps. This is also called *attribution*.

Example 8.2

Think of a social media network with users representing nodes and edges representing "friendships" between them. At $t = 0$, friends of a mutual friend might not be friends. However, at a later time, the mutual friend might introduce them, and they might become friends. Similarly, people make new friends over time and start new friendships online. On the other hand, they might also cut off links — a severed friendship in real life would mean the removal of the edge from the network. This can be viewed as a sequence of static network snapshots at different times.

Algorithms that are used for anomaly detection in dynamic networks slightly differ from that of static networks. Most of these algorithms satisfy the following properties:

1. **Scalability:** In general, network algorithms should be linear or sub-linear in the order of the network size, for the static case. For the dynamic networks, an additional temporal scale requires that the algorithms are easily able to process the 'updates' in the networks over time.
2. **Sensitivity to changes in the structure or context of the network:** The primary property that distinguishes static networks from dynamic networks is the temporal factor that leads to certain changes in the structure of the network. Changes such as adding/removing nodes, edges or labels are widespread. Thus, anomaly detection algorithms for dynamic networks should be sensitive to such changes.
3. **Importance-of-change awareness:** As demonstrated earlier, sensitivity to changes in the structure of the network is important. However, not all changes are important enough to track, as they might also be noise induced over time. Thus, anomaly detection algorithms for the dynamic networks must only track changes in 'important' nodes.

For dynamic networks, the anomaly detection methods can be broadly categorised into four different types:

1. **Feature-based:** Highly similar networks usually share different network features or characteristics such as degree distribution, clustering coefficient, and diameter.
2. **Decomposition-based:** Matrix decomposition of different temporal snapshots for a dynamic network to infer anomalies based on eigenvalues, singular values or top eigenvectors.
3. **Community-based:** Identify anomalies over evolving network clusters in a dynamic network.
4. **Window-based:** Initial temporal snapshots of the network are considered to be normal against which the later instances are studied to spot anomalies.

They differ in the types of 'network summaries' that they use or the type of changes in the networks that they are sensitive to. We shall study them in detail in the following sections.

8.4.2 Feature-based Approaches

As the name suggests, methods in this category utilise 'features' extracted from a network to detect the presence of anomalies over time. The general idea for all feature-based methods remains the same, detailed as follows:

1. Given that we inspect dynamic networks, they are nothing but temporal snapshots of a static network with some changes. Thus, the overall network properties such as degree distribution, diameter, eigenvalues, etc., are similar across time.
2. Thus, the primary motive is to extract a good *summary* of the network. These summaries are extracted using several features, which are used to capture several sensitive changes in the evolving network structure.
3. We then compare consecutive summaries using a 'distance function', that is appropriately chosen for the given problem.
4. If the value of the 'distance' is more than a certain *threshold* set for the given problem, we can say that in that particular timestamp, the network exhibits an anomalous behaviour.

As it can be noted earlier, for a feature-based anomaly detection method on dynamic network, the novelty lies in:

1. The 'network summary' it creates.
2. The distance (or similarity) function it proposes.
3. The definition of the threshold to classify a change as an anomaly.

Some generic approaches include the following:

1. **Maximum common subnetwork (MCS) distance:** In this approach, we calculate the MCS distance between the adjacency matrices or the *2-hop* matrices (2-hop matrix is essentially square of the adjacency matrix) of the consecutive networks.
2. **Error-correcting network-matching distance:** Given any two snapshots of a dynamic network, we can 'superimpose' one network on another and correct errors—edit the nodes, edges, and weights. Thus, this distance metric is calculated by counting the number of operations required to transform one network to the other.
3. **Graph edit distance (GED):** This approach is a simplified version of error-correcting network-matching distance; only topological changes are permitted, i.e., no change in edge weights, etc.

4. **Hamming distance:** Here, we count the number of differences between the adjacency matrices of two networks. This is also called the hamming distance between the two networks.
5. **Variations of edge-weight distances:** In this method, the distance between two nodes is defined by the weight of the edge connecting both of them. Variations in the weight provide a signal of how 'close' two nodes are.
6. **Lambda-distance:** The Lambda-distance is defined as the difference in the whole network spectra, or the top-k eigenvalues of the respective adjacency, 2-hop or Laplacian matrices.
7. **Diameter distance:** In this approach, we calculate the diameter distance, which is defined as the difference in the network diameters (the diameter of a network is the longest of the shortest distances between any two nodes).

Now, we shall explore in detail, one of the early methods for anomaly detection, developed by Akoglu and Faloutsos (2010). The intuition behind this algorithm is simple—*a node is anomalous if its current 'behaviour' deviates from its 'normal behaviour' based on past timestamps*. We proceed to answer two essential questions:

1. At what point in time, do several nodes change their behaviours significantly?
2. Is it possible to characterise those nodes that change their behaviour frequently?

This algorithm also employs the method of ego-centric networks. Each node is characterised on the basis of its egonet. The 12 features used to describe this characteristic (extracted from the egonet) are: in-degree, out-degree, in-weight (sum of weights of in-edges), out-weight (sum of weights of out-edges), number of neighbours (total degree), number of reciprocal neighbours (strongly connected), number of triangles (number of neighbour pairs who share an edge), average in-weight, average out-weight, maximum in-weight, maximum out-weight, and maximum weight ratio on the reciprocated edges in the egonet.

1. Each network is represented as an $N \times F$ feature matrix (N = number of nodes, F = number of features), and we have T such networks (T = total timestamps). Therefore, data is in the form of a $T \times N \times F$ three-dimensional tensor.
2. Extract a slice of the tensor for a feature in the shape of $T \times N \times 1$.
3. Define W, a window size. For each pair of nodes, calculate the correlation coefficient (Pearson's) between their time-series vectors over the window of size W.
4. Now, the window slides by 1 time-tick, and the correlation coefficients are calculated for the next window of W time-ticks. For every window, we shall get an $N \times N$ (every pair of nodes) matrix of correlation coefficients. This matrix signifies how every node in the network is correlated with every other node in the network. The total number of such matrices is $T - W$ (call it C). Therefore, we now have a three-dimensional matrix of order $N \times N \times C$.
5. Then, we extract the principal eigenvector for all of the C matrices of size $N \times N$. In this vector, the value corresponding to each node can be called the "activity"

of this node. This activity will be higher for nodes that are more correlated to the majority nodes. Such an eigenvector is called 'eigen-behaviour' of all nodes in the network as a whole. 'Normal activity' is defined as the average value of a vertex in all the principal eigenvectors.

6. Compute 'typical-eigen-behaviour' for $u(t)$, which is the principal eigenvector at time, $t - r(t-1)$ (the average of principal eigenvectors or the normal activity vector).
7. Now, the timestamp seems to have been lost. How do we know at which timestamp the anomaly occurs?

   The answer is simple—the change is $Z = (1 - u^T r)$, where u is the eigenvector at time T and r is the normal eigen-behaviour vector. If Z is zero, that means $u(t) = r(t-1)$, i.e., no anomalous behaviour. If Z is 1, which means $u(t)$ and $r(t-1)$ are very different (at angle 90°), and it is anomalous.

Figure 8.11 demonstrates this method.

8.4.3 Decomposition-based Approaches

So far, we have seen that certain types of curated features help in analysing a dynamic network by studying the differences in features over time. A spike in these differences would be characterised as an anomaly. However, there are several decision points one has to make. *Which features to use*? *Which distance measure to use*? *Should we delete or add more features?* Such general decision issues with feature-based methods lead to the development of other alternatives.

The primary idea of decomposition-based methods is as follows. Decomposition methods (SVD) are a very powerful way to extract summaries of networks. If we can

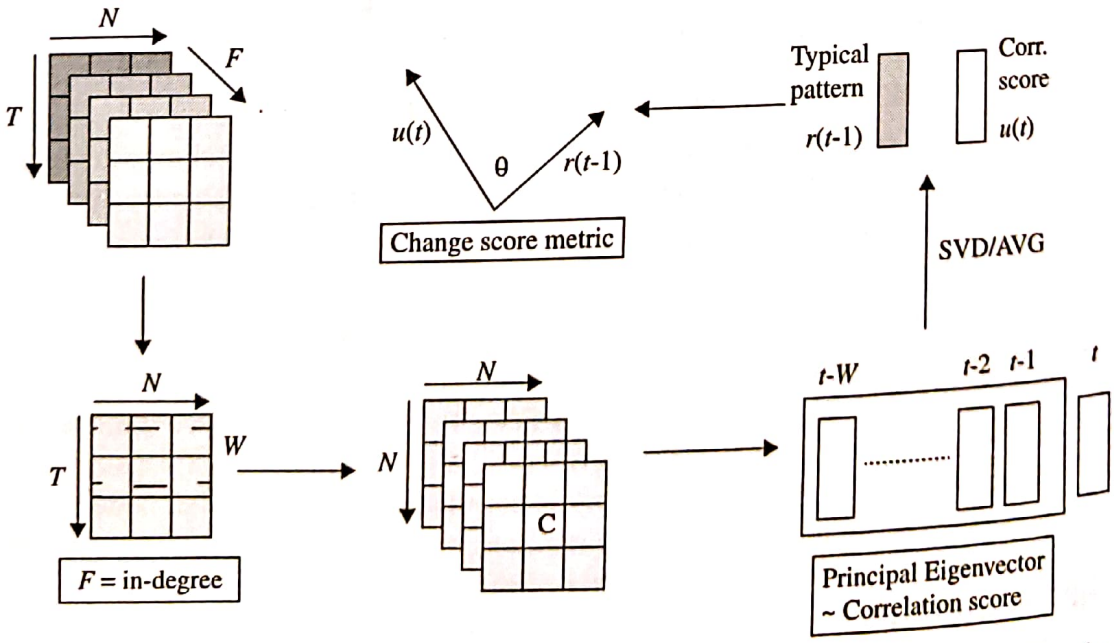


FIGURE 8.11. Framework of feature-based anomaly detection in dynamic networks.

extract tensor or matrix decompositions of a dynamic network, we would be able to analyse and interpret signals from the selected eigen-vectors or singular vectors.

Thus, decomposition-based methods require the following information for detecting anomalies:

1. **An indication of 'normal' behaviour:** In order to detect anomalies, these methods first characterise normal behaviour in terms of the decompositions extracted from the matrix representation of networks.
2. **Similarity measure:** A similarity metric is required that would be able to help in distinguishing normal behaviour from the anomalous ones.

IdÉ and Kashima (2004) provided a simple approach for a decomposition-based method, given in Figure 8.12. They extracted principal eigenvectors from the adjacency matrix of each snapshot of the dynamic network. These vectors are known as **activity vectors**. The activity vectors of different timesteps are combined, on which SVD is applied to extract a 'typical' activity vector. In an online manner, the typical activity vector is updated after every timestamp. Given a typical activity vector, it can be compared with an incoming activity vector using the cosine similarity metric. Deviations or anomalies are estimated based on the distance between a typical vector and the incoming activity vector.

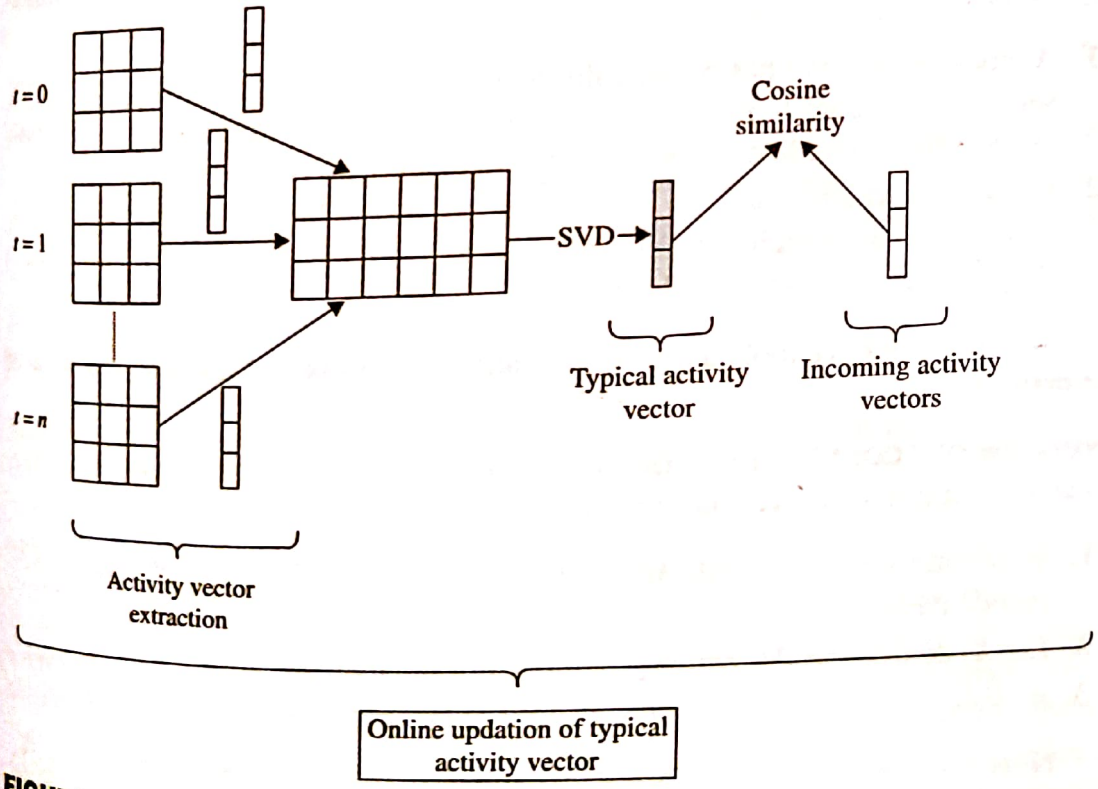


FIGURE 8.12. Decomposition-based dynamic anomaly detection by IdÉ and Kashima (2004). Since this is an online algorithm, after every time an incoming new activity vector is compared with the typical activity vector, the typical activity vector is updated.

8.4.4 Community-based Approaches

In the community-based anomaly detection, we focus on how the membership of nodes in different communities changes over the course of time. The idea behind doing this is that most of the times, members of the same community in a network tend to behave similarly. Given the snapshots of the network at different timestamps, we want to flag nodes that lead to a significant change in the structural properties of the network, in the context of community membership. Therefore, in this class of methods, the tasks of community detection and outlier detection occur simultaneously.

In this category of methods, we shall look at `ECOutliers`, proposed by Gupta et al. (2012) in detail.

Detecting Evolutionary Community Outliers

How often have you seen students in your class, take courses that are different from their peers? Have you noticed how most of the families living in the same housing society tend to send their children to a few selected schools? Usually, members of the same community tend to behave similarly with time. However, some nodes in the network might change their behaviour significantly from the average behaviour of the community. These nodes are referred to as 'Evolutionary Community Outliers' by Gupta et al. (2012). Furthermore, the authors also proposed a method for detecting these outliers, called `ECOutliers`. Some more examples of evolutionary community outliers can be:

1. A stockbroker who has historically had a portfolio rich in IT company stocks, suddenly switches to petrochemical-based companies. However, the rest of the stock brokers that had a similar portfolio display no change in their trades.
2. Reliance, which used to focus on its petrochemical-based businesses, announced the launch of Reliance Jio, its telecom venture. None of its historical competitors made any similar moves.

Now that we have looked at the motivation, let us take a look at an overview of the method.

Overview of `ECOutliers`. Before we proceed further, let us define some notational conventions that we shall follow:

1. M denotes a matrix, while $M_{i,.}$ and $M_{.,j}$ denote the i^{th} row and j^{th} column of M, respectively.
2. Let X_i denote the i^{th} temporal snapshot of the network.
3. $a \cdot b$ denotes the inner product or dot product between the vectors a and b.

Now, let us say we have a sequence of snapshots of the network, $X_1, X_2, \ldots, X_n$. In the scope of this work, Gupta et al. (2012) focused on detecting outliers between any pair of snapshots, X_i and X_j. For the sake of simplicity, let us refer to the two snapshots as X_1 and X_2. We can use any state-of-the-art community detection algorithm to mark the communities of the nodes in each of the snapshots. Let us say, the

number of communities in X_1 and X_2 is K_1 and K_2, respectively. These community assignments also lead us to *community belongingness matrices* for X_1 and X_2. These matrices P, Q (for X_1, X_2, respectively) give us the likelihood of a node belonging to a community. Clearly, $P \in [0,1]^{N \times K_1}$ and $Q \in [0,1]^{N \times K_2}$. Additionally, the sum of the elements in each row of both P and Q would be 1. Hence, for the o^{th} row of the matrices:

$$\sum_{i=1}^{K_1} P_{oi} = 1 \quad \sum_{j=1}^{K_2} Q_{oj} = 1$$

where o represents an object/node in the network.

Now that we are familiar with the objects required to describe each snapshot, let us talk about finding a "correspondence" between them. Essentially, what we want to try and do, is to match communities in both the snapshots. In order to do this, the authors defined a *correspondence matrix*, denoted by S. They proposed a soft correspondence, i.e., each community in X_1 is matched with every community in X_2, but with a different weight. Thus, $S \in [0,1]^{K_1 \times K_2}$ and for any o^{th} row in the matrix,

$$\sum_{j=1}^{K_2} S_{oj} = 1$$

Note that we do not know the final correspondence matrix beforehand. We learn the optimal correspondence matrix through the algorithm so that it gives us the best matching between the communities in the two snapshots.

Ideally, what we expect is that the community structure in the network would not change by a lot. In order for this to happen, the node must not behave very differently from the rest of the members of the same community. Hence, to quantify the anomalous behaviour of node-community pairs, we define an *outlierness matrix*, denoted by A. The dimension of A is $N \times K_2$, and the $(o,j)^{\text{th}}$ element gives the outlier score for node o and community j in X_2.

With this, we have setup all the quantities that we need in order to proceed. Now, we take a look at what the algorithm tries to do. The algorithm mainly focuses on *community matching* rather than anomaly detection. In order to do so, we need to estimate the optimal correspondence matrix S so that the difference between Q and ($P.S.$) is minimised. Note that S essentially acts as a linear transformation, mapping the two belongingness matrices. However, some nodes in the network might be outliers, and no matter how well we try to estimate S, these nodes will create a problem by biasing the algorithm. Hence, the authors developed a framework, where they could incorporate the estimation of the outlier score of a node with its community through the outlierness matrix A, while penalising the contribution of a node to the estimation of S, depending on its outlier score. At the end, an object-community pair (o,j) is an `ECOutlier` if the change from P_{oi} to Q_{oj} is very different than the change in belongingness in the other nodes between communities X_i and X_j. We now look at the joint framework for estimating S and A in more detail.

Joint framework for estimating S and A. `ECOutlier` is framed as the following optimisation problem:

$$
\begin{aligned}
\min_{S,A} \quad & \sum_{o=1}^{N}\sum_{j=1}^{K_2} \log\left(\frac{1}{A_{oj}}\right)(Q_{oj} - P_{o,}.S_{.,j})^2 \\
\text{s.t.} \quad & S_{ij} \geq 0 && \forall i = 1,2,\ldots,K_1, \forall j = 1,2,\ldots,K_2 \\
& \sum_{j=0}^{K_2} S_{ij} = 1 && \forall i = 1,2,\ldots,K_2 \\
& 0 \leq A_{ij} \leq 1 && \forall i = 1,2,\ldots,K_1, \forall j = 1,2,\ldots,K_2 \\
& \sum_{i=1}^{N}\sum_{j=1}^{K_2} Aij \leq \mu
\end{aligned}
\tag{8.8}
$$

Let us now go over all the parts of this optimisation problem to understand why is it framed this way:

1. $\min_{S,A} \quad \sum_{o=1}^{N}\sum_{j=1}^{K_2} \log\left(\frac{1}{A_{oj}}\right)(Q_{oj} - P_{o,}.S_{.,j})^2$

   The $(Q_{oj} - P_{o,}.S_{.,j})^2$ term essentially denotes the square of the difference between each element of Q and $(P.S)$. So what we are trying to do is to minimise the difference between Q and $(P.S)$, which is what we need to do for community matching. At the same time, we have the term $\log\left(\frac{1}{A_{oj}}\right)$. This is to reduce the influence of a node-community pair, if their anomaly score is high. The higher A_{oj}, the lower the value of the term, and thus, the smaller the contribution of pair (o,j) to the difference.

2. $S_{ij} \geq 0 \quad \forall i = 1,2,\ldots,K_1 \quad \forall j = 1,2,\ldots,K_2$

   S_{ij} is an element of the correspondence matrix. It denotes the soft correspondence between community i in snapshot X_1 and community j in X_2. One can also think of it as the extent to which community i in X_1 matches with community j in X_2. Hence, these values must be greater than or equal to 0.

3. $\sum_{j=0}^{K_2} S_{ij} = 1 \quad \forall i = 1,2,\ldots,K_2$

   $\sum_{j=0}^{K_2} S_{ij}$ is the sum of the i^{th} row of the correspondence matrix. It denotes the sum of the degree of matches of the i^{th} community in X_1 to all the communities in X_2. Therefore, intuitively, this sum is 1.

4. $0 \leq A_{ij} \leq 1 \quad \forall i = 1,2,\ldots,K_1 \quad \forall j = 1,2,\ldots,K_2$

   A_{ij} is the outlierness score of node i and community j. It represents the probability that the pair is an outlier. So, the constraint follows intuitively.

5. $\sum_{i=1}^{N}\sum_{j=1}^{K_2} Aij \leq \mu$

$\sum_{i=1}^{N}\sum_{j=1}^{K_2} Aij$ represents the sum of all elements in the outlierness score matrix, A. If we observe the objective function again, we realise that we have all the A_{ij}'s in the denominator. So, a trivial way to make the objective go to a minimum would be to allow A_{ij}'s to assume arbitrarily large values, leading the entire sum to $-\infty$. However, this solution would not at all be useful to us as it would label all the nodes as outliers. Hence, to counter this, we impose a constraint on the sum of all the outlierness scores, and cap it by a value μ. μ is the estimated sum of outlierness in a snapshot, and its final value will also be learnt as we estimate A.

Now that we have discussed the algorithm as an optimisation problem, how do we solve it? We turn to the method of Lagrange Multipliers for solving this problem. (We shall not go into the details of this concept. If you would like to learn more about this technique, refer to additional resources at the end of this chapter.) However, by using this technique, we arrive at the following optimisation problem:

$$\begin{aligned} \min_{S,A} \quad & \sum_{o=1}^{N}\sum_{j=1}^{K_2} \log\left(\frac{1}{A_{oj}}\right)(Q_{oj} - P_{o,}.S_{.,j})^2 + \sum_{i=1}^{K_1}\beta_i\left(\sum_{j=0}^{K_2} S_{ij} - 1\right) + \gamma\left(\sum_{i=1}^{N}\sum_{j=1}^{K_2} Aij - \mu\right) \\ \text{s.t.} \quad & S_{ij} \geq 0 \quad \forall i = 1, 2, \ldots, K_1, \forall j = 1, 2, \ldots, K_2 \\ & 0 \leq A_{ij} \leq 1 \quad \forall i = 1, 2, \ldots, K_1, \forall j = 1, 2, \ldots, K_2 \end{aligned} \tag{8.9}$$

In order to obtain the minimal of this objective function, we take its partial derivative with respect to A_{oj} and set it to 0, leading us to the following expression:

$$A_{oj} = \frac{\left(Q_{oj} - P_{o}.S_{.j}\right)^2}{\gamma} \text{ and } \gamma = \sum_{o=1}^{N}\sum_{j=1}^{K_2}\frac{\left(Q_{oj} - P_{o}.S_{ij}\right)^2}{\mu} \tag{8.10}$$

Using Equations 8.9 and 8.10, we can define an update rule for A_{oj} as,

$$A_{oj} = \frac{\left(Q_{oj} - P_{o}.S_{.j}\right)^2 \mu}{\sum_{o'=1}^{N}\sum_{j'=1}^{K_2}\left(Q_{o'j'} - P_{o'.} \cdot S_{.j'}\right)^2} \tag{8.11}$$

Similarly, taking the partial derivative of the objective function in Equation 8.9 with respect to S_{ij} and setting it to 0, we obtain:

$$\sum_{o'=1}^{N}\left[2\log\left(\frac{1}{A_{o'j}}\right)\left(Q_{o'j}-P_{o'}.S_{.j}\right)\left(-P_{o'i}\right)\right]+\beta_i=0 \tag{8.12}$$

$$\Longrightarrow S_{ij}=\frac{\sum_{o'=1}^{N}2\log\left(\frac{1}{A_{o'j}}\right)P_{o'i}\left[Q_{o'j}-\sum_{\substack{K=1\\K\neq i}}^{K_1}P_{o'k}S_{kj}\right]-\beta_i}{\sum_{o'=1}^{N}2\log\left(\frac{1}{A_{o'j}}\right)P_{o'i}^2} \tag{8.13}$$

Equation 8.13 is used as the update rule for S_{ij}. But how do we make use of these update rules? We use these update rules to iteratively update A and S until a point of convergence is reached. This procedure is described more comprehensively in Algorithm 8.3.

Algorithm 8.3 `ECOutliers`

Input: Belongingness matrices, P, Q
Output: Estimates of correspondence matrix, S and outlierness score matrix, A

1 Initialise μ to 1
2 Initialise $S_{ij} \leftarrow \frac{1}{K_2} \forall i,j$
3 Initialise $A_{ij} \leftarrow \frac{1}{NK_2} \forall i,j$
4 **while** *not converged* **do**
5 |Update A using Equation 8.11
6 |Update S using Equation 8.13
7 **end**
8 $\mu \Leftarrow \frac{\sum_{o'=1}^{N}\sum_{j'=1}^{K_2}\left(Q_{o'j'}-P_{o'}.S_{.j'}\right)^2}{\max_{o',j'}\left(Q_{o'j'}^2\right)}$
9 Repeat Steps 2 to 6

As we can see from Algorithm 8.3, it computes the correspondence matrix S and the outlier matrix A simultaneously. The algorithm converges when the change in the objective function is less than some constant ϵ. However, there is still one question that is unanswered. How do we arrive at an update value for μ?

Updating the expected total outlierness (μ). As we can see, initially, μ is set to 1. However, such a small value of μ does not leave much room for the outlierness scores A_{oj} to be distributed, making it difficult to distinguish between the outlier and the non-outlier node-community pairs. One way to resolve this is to update the value

of μ to a quantity that allows A_{oj} values to grow as large as possible, while remaining within the constraints. This is achieved by dividing the error at the end of the first-pass by the maximum value in the belongingness matrix Q. This results in an increase in the value of μ in the second-pass, giving more varied values of A_{oj} resulting in the easier outlier filtering.

There is still one question unanswered. Algorithm gives us outlier scores and community matching. However, how do we single out the outliers from these matrices?

Identifying the outliers. At the end of Algorithm 8.3, we obtain outlier scores for every node with respect to every community in the network. There are multiple ways to proceed from here:

1. The entries of the outlier matrix A could directly be used for filtering outliers with respect to each community using some sort of thresholding on the outlier scores. For instance, nodes having outlier scores above $\frac{\mu}{N}$ could be considered as outliers.
2. One might also aggregate the outlier scores for a node across all the communities. This can be done in multiple ways, such as taking mean of outlier scores across all the communities ($\hat{A}_o = \frac{\sum_{j=1}^{K_2} A_{oj}}{K_2}$), or a weighted mean, weighted by the size of the communities ($\hat{A}_o = \frac{\sum_{j=1}^{K_2} |j| A_{oj}}{K_2}$, where $|j|$ denotes the size of community j), or the maximum outlier value for a node ($\hat{A}_o = \max_j A_{oj}$). Depending on the application, different options can be explored.
3. When enlisting the outliers in a given snapshot, one might also want to consider the overall activity of the node during the snapshot. If a stock broker has 100 transactions in a day but a moderate outlier score, she might still be more interesting than a broker with just 10 transactions but a slightly north of moderate outlier score.

8.4.5 Window-based Approaches

So far, we have studied three approaches for anomaly detection from dynamic networks. However, for all these approaches, the anomaly prediction is made based on all the snapshots that have occurred previously. In the *window-based methods*, we bind our snapshot history to a certain *window* and categorise the *normal* behaviour in that particular window.

Window Size

Given a sequence of objects x, we can define a window as a subset of x, consisting of elements that occur sequentially in x. The number of elements in this subset is the window size.

Hop Length

The number of elements that are skipped between the starting points of two consecutive windows is called the hop length.

Example 8.3 Let a sequence of numbers $x = 1, 2, 3, 4, 5, 6, 7, 8, 9$. A window of window size 3 is $1, 2, 3$. If we have a hop length of 2, the next window in our sliding window approach would be $3, 4, 5$. Here, we can notice that 1 and 2 are 'hopped' over by the next window.

One such window-based approach was provided by Priebe et al. (2005). Their primary idea is to spot anomalies by identifying snapshots that have unusually high connectivity, as compared to the previous time steps. This is shown in Figure 8.13.

Their approach can be summarised as follows:

1. A *local statistic*, which is a statistic that outputs a certain value for a selected window. The maximum value of these local statistics is called *scan statistic*.
2. Given an email network, nodes denote users, and directed edges are constructed to determine who sent emails and to whom.
3. For each window, the local statistic is calculated as follows:
   - Given $k = 0, 1, 2$, the k-step neighbourhood is extracted.
   - The *number of edges* in this k-step neighbourhood is the local statistic for this window.

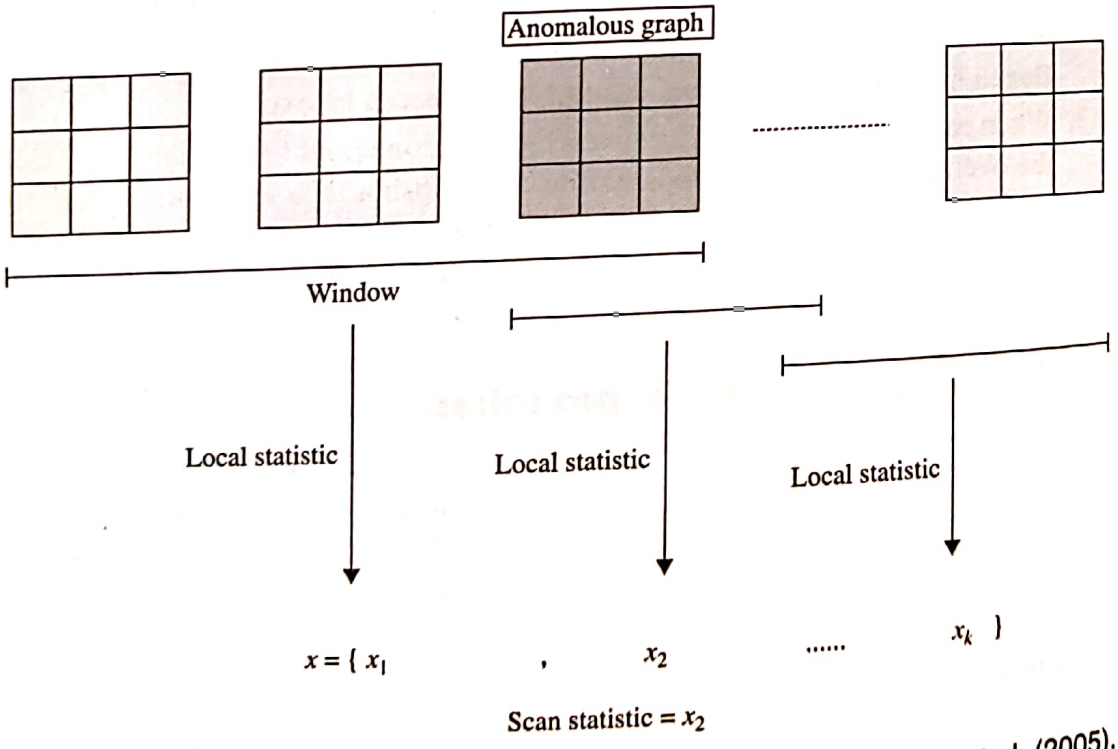


FIGURE 8.13. Window-based dynamic anomaly detection by Priebe et al. (2005). The scan statistic is x_2. If $x_2 > \tau$, we would claim that an anomaly is present.

4. If the scan statistic exceeds a certain threshold τ, the given corresponding window is determined to be anomalous.

8.5 CHAPTER SUMMARY

In this chapter, we have looked at the problem of anomaly detection from the perspective of network data sets. It has been seen that network-based anomaly detection can capture inter-dependencies among various entities in the network, essential for an in-depth analysis. Anomaly detection in a network setting comes with a lot of challenges that need to be considered while proposing a novel strategy for the same, namely, lack of labelled data, robust algorithms, and large network-based datasets. Following this, we have looked at numerous existing methods for network-based anomaly detection. A different set of algorithms have been explored for different types of networks: (a) plain static networks (structure and community based), (b) attributed static networks (structure and community based), and (c) plain dynamic networks (feature, decomposition, community, and window based). Our discussion in this chapter will motivate the readers to explore the development of robust algorithms to tackle an adversarial attack. Along with this, given that the real-world networks available today grow at an exponential rate, scalable anomaly detection algorithms are required. In Chapter 10, we shall present other applications such as fraud detection, which can also be solved using anomaly detection methods.

ADDITIONAL RESOURCES

- **Important papers/blogs/reports**
  - A thorough survey paper on network-based anomaly detection: Akoglu et al. (2015), Ranshous et al. (2015), Pourhabibi et al. (2020)
  - Khan Academy Tutorial on Lagrange Multipliers: https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/constrained-optimization/a/lagrange-multipliers-single-constraint (Last accessed on 10 August 2020)
  - Video lecture on `ECOutliers` by Gupta et al. (2012): https://www.youtube.com/watch?v=husp0JNc6gI (Last accessed on 10 August 2020)
  - Anomaly detection in dynamic graphs using MIDAS: https://lionbridge.ai/articles/introducing-midas-a-new-baseline-for-anomaly-detection-in-graphs/
  - Blog: https://towardsdatascience.com/anomaly-detection-in-dynamic-graphs-using-midas-e4f8d0b1db45 (Last accessed on 10 November 2020)
  - AddGraph: Zheng et al. (2019)
- **Source codes**
  - A good toolkit for anomaly detection in Python: Zhao et al. (2019)
  - Codes for anomaly detection methods: https://paperswithcode.com/task/anomaly-detection (Last accessed on 10 November 2020)

EXERCISES

Phrase the Problem

Given below are some scenarios. Phrase an anomaly detection problem in each case. Your problem statement should properly define an anomalous entity and the network that you have modelled from this scenario, whether it is static or dynamic, etc. You can assume some extra information that is needed for phrasing your problem, but it should be reasonable and in sync with the scenario described.

1. You have logs about the requests and assignments of a major cab-providing service, along with their timestamps.
2. You have information about a set of "influencer" profiles on a significant social network platform. You know what they write about, who they follow, and are followed by, etc.
3. You have information about citations of a set of scientific publications.
4. You have the data on different dates about the complete evolution of an online discussion forum, from things such as which users had an active account on that day to who participated with whom in which discussion.
5. You have information about the courses taken and research topics followed by the students in a university, as well as different majors that they are registered under.

Objective Type Questions

1. Abnormality detection is data cleaning, i.e., removal of erroneous values or noise from data for better prepossessing to learn accurate models. (True/False)
2. Which of the following can be considered as global measures to be used as a feature in anomaly detection:
   (a) Number of connected components
   (b) Minimum spanning tree
   (c) Average node degree
   (d) Options (a) and (b)
   (e) All of the above
3. In an undirected network, the stationary probability of node u is dependent on the initial node configuration and inversely proportional to the degree $deg(u)$. (True/False)

Short Answer Questions

1. What is the difference between event and change detection in a network series G to detect anomalies?
2. Which one kind of change cannot be detected by anomaly detecting techniques?
3. Explain in detail the challenges associated with anomaly detection and attribution.
4. Given the egonet, the challenge is which feature to look at from the possible network-based measures that can be extracted as egonet features. Justify whether it is better to analyse the egonet features in pair or as union.
5. How would you describe a dynamic-network anomaly detection problem? What are all the properties desired to be there in the algorithms?
6. In the structural-based approaches for anomaly detection, we find the unusual substructures by using MDL (minimum description length). What is the major objective behind this principle, and how does it find the unusual substructures?
7. What are the problems on detecting anomaly using community-based detection methods? How can those problems be resolved?

Long Answer Questions

1. You have an anomaly detection system that gives 95% accuracy. Would you be satisfied with this performance? Why?
2. State and explain some practical data-specific challenges that arise in anomaly detection problems.
3. Let us say that you are trying to detect outliers in a community setting. Does it make sense to study community detection and outlier detection separately in this case? Why or why not?
4. Let us say that you have a system that can detect anomalies in any given scenario, with a false-positive rate of 50% and a false-negative rate of 15%. Would you be happy with such a system? If not, can you explain with a scenario where such a system could be more destructive than helpful? What if the values of false-positive and false-negative rates were interchanged? Can you again describe your disagreement with a scenario?
5. You read a published paper that says that the number of edges connected to a node in a network is directly proportional to the age of the node. This study was published in 2005. Would you blindly follow this study in 2020? Justify your answer with reason.
6. What are the evaluation strategies for a network-based anomaly detection approaches?
7. Probabilistic methods are used to construct a model of what is considered as "normal" or expected by using probability theory distributions and scans statistics. Deviation from the model is flagged as anomalous. How can you create such a model, and determine the methods for detecting anomalies?
8. Formulate a model for detecting non-crashing software bugs in a given static attributed network.
9. Fraudsters often evade detection using camouflage, by adding reviews or follows with honest targets so that they look "normal". Given a bipartite network of users and the products that they review, or followers and followees, how can we detect fake reviews or follows? Consider a set of m users $U = \{u_1, \ldots, u_m\}$ and n objects $W = \{w_1, \ldots, w_n\}$ connected according to a bipartite network $G = (U \cup W, E)$, E is the set of edges. We can consider the objects to be followees on Twitter or products on Amazon.

BIBLIOGRAPHY

Akoglu, L., and Faloutsos, C. (2010). Event detection in time series of mobile communication graphs. *Army Science Conference*, vol. 1. Orlando, Florida, United States.

Akoglu L., McGlohon M., Faloutsos C. (2010) oddball: Spotting Anomalies in Weighted Graphs. In: Zaki M.J., Yu J.X., Ravindran B., Pudi V. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2010. *Lecture Notes in Computer Science*, vol. 6119. Springer, Berlin, Heidelberg.

Akoglu, L., Tong, H., and Koutra, D. (2015). Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3), 626–688.

Brin, S., and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Comp. Net. ISDN Sys.*, 30(1), 107–117.

Chakrabarti D. (2004) AutoPart: Parameter-Free Graph Partitioning and Outlier Detection. In: Boulicaut JF., Esposito F., Giannotti F., Pedreschi D. (eds) Knowledge Discovery in Databases: *PKDD 2004*. PKDD 2004. *Lecture Notes in Computer Science*, vol: 3202. Springer, Berlin, Heidelberg.