Chapter 12

Outlier Detection in Graphs and Networks

"In nature, we never see anything isolated, but everything in connection with something else which is before it, beside it, under it and over it."– Johann Wolfgang von Goethe

12.1 Introduction

Graphs represent one of the most powerful and general forms of data representation. These structures are used to express diverse data, ranging from multidimensional entity-relation graphs, the Web, social networks, communication networks, and biological and chemical compounds. Broadly speaking, two types of graphs arise in real domains:

- The data may contain many small graphs, drawn over a small base domain of labeled nodes. Some examples of this scenario include chemical and biological compounds. The labels correspond to the chemical elements, which may be repeated within the same object or different objects. The repetition of node labels causes a serious computational challenge in such applications, which is referred to as *graph isomorphism*. Individual graph objects are defined as outliers based on the model of normal graph objects in the database. Therefore, outlier scores are associated with entire graphs as well.

- The data may be represented as a single large graph. Examples include the Web, social networks, and information networks. In most cases, such as the Web and social networks, the nodes correspond to distinct identifiers such as Uniform Resource Locators (URLs), actors, or IP addresses. In some cases, it is also possible for node labels to be repeated. In this setting, outlier scores are defined on the structural elements (e.g., nodes, edges, subgraphs) of the single large graph.

Many other natural variations of the aforementioned scenarios arise, such as the extraction of multiple small graphs from a larger network. An example of such a scenario is a bibliographic network, in which a publication may be represented as a small graph over a larger bibliographic co-authorship network. In this setting, the smaller graphs may be defined as outliers based on their linkage relationships. As discussed in section 12.3.2.4, the methods for outlier analysis in a single large graph can be easily generalized to this case. These different settings require very different models. For example, in the case of small graphs, a single object may be represented as an outlier. However, in large graphs, the outliers are defined as portions of the network, which might be nodes, edges, or even subgraphs.

In temporal settings, outliers typically correspond to structural changes in large-scale networks like the Web, social or communication networks. Structural changes may be modeled in terms of communities, shortest paths, or other local structural properties. Temporal graphs represent one of the most challenging cases for outlier analysis, because of the many different ways in which outliers may be defined. The following phrase from Chapter 1 is restated here: "*The more complex the data is, the more the analyst has to make prior inferences of what is considered normal for modeling purposes.*" For example, in a temporal network, an outlier node could be a node with unusually high degree, unusual connectivity structure, unusually *changing* degree, unusually *changing* community structure, unusually *changing* distances to other nodes, or unusual relationships of node content to linkage structure. There is virtually an unlimited number of different ways in which outliers could be defined. Even within the context of a specific outlier type such as a node outlier, the appropriate model of regularity could be based on its degree, neighbor set, edge-weight distribution, and so on. Therefore, even the *definition* of an outlier in a very complex data type, such as a graph, might lend itself to a bewildering number of possibilities.

In such scenarios, it is important to define and model the outliers from an application-specific perspective, because no uniform definition exists and a specific application might provide better guidance. For example, in a spam detection application, the degree distributions of nodes can provide insights about outliers. In a network de-noising application, the linkage connectivity structure can be used to determine outlier links. This type of domain-specific knowledge can be considered a mild form of supervision, since it is often created based on prior data-centric experiences. This chapter will focus on a number of specific definitions of network outliers because of their ubiquity in many tasks. Nevertheless, the methods discussed are certainly not exhaustive and there might be numerous settings in which other definitions might be more appropriate.

This chapter will study the following aspects of outlier detection in graphs and networks:

- The problem of outlier detection in many small graphs will be introduced. Such graphs occur commonly in the domain of chemical data, and are therefore useful in determining unusual structural combinations.

- Different models for outlier analysis will be studied in the context of a single large graph. These cases arise commonly in the context of the Web, communication networks, and social networks. However, the complexity of a large network allows the definition of outliers in a variety of interesting ways. This setting is particularly important because of the increasing prevalence of very large graphs in real-world settings.

- The issue of temporal outlier detection in graphs will be studied. This area is intimately connected to the topic of *evolutionary network analysis* [14]. The goal is to study significant local and global structural changes in graphs that are abrupt and unexpected.

Different variations of the aforementioned scenarios will be addressed, such as the incorporation of node content or domain knowledge into the outlier analysis process.

This chapter is organized as follows. Section 12.2 addresses the problem of outlier detection in many small graphs. Section 12.3 discusses the problem of outlier detection on a single large graph. The case of many small graphs super-imposed on a single large graph will also be discussed within this section. Methods for incorporating node content in outlier analysis are discussed in section 12.4. The problem of change analysis and outlier detection in temporal graphs is discussed in section 12.5. The conclusions and summary are presented in section 12.6.

12.2 Outlier Detection in Many Small Graphs

An interesting case for outlier analysis arises when the data contains many small graphs. Some examples include chemical compounds, biological networks, XML graphs, Resource Description Framework (RDF) graphs, and entity-relation graphs. In some of these domains, the graphs could become quite large, although the graphs are of relatively modest size in most cases. A complicating factor in these domains is that the labels on nodes may typically be repeated, as a result of which the matching can be performed in a variety of different ways. This is referred to as *the challenge of isomorphism*. Therefore, similarity computations can sometimes be difficult. A discussion of these issues is provided in [558].

The problem of outlier detection in such cases is similar to multidimensional point anomaly detection, where each graph object is treated as an individual point. In this context, the easiest class of models to generalize is that of proximity-based models. This is because the problems of clustering and similarity search have been widely studied in domains such as chemical compound mining and XML graph analysis [15, 558]. In the case of clustering methods [15], the clusters are defined as sets of graphs that overlap with the frequent subgraph patterns in the underlying data set. Since outliers are defined in a complimentary way to clusters, they are defined as graphs that do not overlap well with the frequent subgraph patterns in the data. Thus, a natural approach for outlier detection in such scenarios is to determine the frequent subgraphs in the underlying data, and determine those graphs which do not contain these frequent patterns. This approach is analogous to determining the outliers in transaction data with the use of pattern mining. Outliers can be defined as graphs which do not contain a significant number of frequent subgraph patterns. Therefore, an approach analogous to that discussed in section 8.5.1 of Chapter 8 for outlier detection in transaction data [254] can also be used for the case of graphs. As in the case of transaction data, support-based quantifications can be used in this case, except that the support is defined on subgraph frequent patterns, rather than transaction frequent patterns.

The use of a k-nearest neighbor outlier detection algorithm is also a viable alternative in this case, because of a wide variety of available algorithms for similarity search in graphs [558]. Numerous similarity functions are commonly used such as the graph edit distance, largest common subgraph, largest matching node set, to perform the similarity search. A detailed discussion of commonly used similarity and distance functions in the graph domain may be found in [33] (Chapter 17). While these methods correspond to generic extensions of multidimensional outlier detection algorithms, not much work has been specifically targeted towards outlier detection in this domain.

12.2.1 Leveraging Graph Kernels

The design of effective similarity functions is helpful not only for distance-based outlier detectors but also for using other linear models such as PCA or one-class support-vector machines. For this purpose, we need a special type of similarity function, which is also referred to as a *graph kernel.* Two well-known kernels referred to as the random-walk kernel and the shortest-path kernel are discussed in [33]. For a database containing N graphs, one can construct an $N \times N$ kernel similarity matrix S. This similarity matrix can be used in two different ways to detect outliers:

1. One can use the one-class SVM of section 3.4 in Chapter 3 to create a model of normal graphs. This model can be used to compute the outlier scores of the graphs.

2. One can use soft kernel PCA, which is discussed in section 3.3.8 of Chapter 3. The Mahalanobis method can be used in conjunction with the resulting embedding in order to score the data points. The extension of the Mahalanobis method to arbitrary data types is discussed in section 3.3.8.3.

Graph kernel methods can also be used for supervised outlier detection with the use of kernel support-vector machines.

12.3 Outlier Detection in a Single Large Graph

In large graphs, unusual structural characteristics can be used to define outliers in different regions of the network. Such outliers can be defined in a variety of ways, such as node outliers, linkage outliers, or subgraph outliers. Each of these different types of outliers will be discussed in detail. Throughout this section, it will be assumed that a single large network or graph is available for analysis. This graph is denoted by $G = (N, A)$. Here, N denotes the set of nodes, and A denotes the adjacency matrix of the set of edges. It is assumed that the number of vertices in N is n, and the number of edges in A is m. Therefore, A is an $n \times n$ matrix, which contains $m < n^2$ nonzero entries. In most settings, the matrix G is sparse, and therefore we have $m \ll n^2$. Unless otherwise mentioned, it will be assumed that the graph G is undirected; however, some of the following methods can also be generalized to directed graphs.

12.3.1 Node Outliers

Node outliers can be defined in a network in a wide variety of ways. The key is to extract features from the neighborhood of a node, and then define the outliers in terms of these features. The work in [41] defines a set of features that are extracted from the 1-step neighborhood (also known as *egonet*) of node i. The features are as follows:

- (Node Feature n_i) The node feature is the number of nodes in the 1-step neighborhood of node i (which is the same as its degree).

- (Edge Feature e_i) The edge feature is the number of edges between all nodes in the 1-step neighborhood of node i.

- (Weight Feature w_i) For weighted graphs, the weight feature comprises the total weight of all edges in the 1-step neighborhood of node i.

- (Spectral Feature λ_i) The spectral feature is the principal eigenvalue of the weighted subgraph in the 1-step neighborhood of node i.

These features can then be organized into carefully chosen pairs, which are shown to obey a number of interesting power laws. Data points that deviate from these power laws are flagged as outliers. Some examples of useful pairs of features for anomaly detection are as follows:

- **Node feature versus edge feature**: Extremely dense neighborhoods of a node correspond to near-cliques, whereas extremely sparse neighborhoods of a node are stars. By plotting the relationship between the node and edge features, it is possible to detect near cliques and stars. In general, the *expected relationship* between these features is $e_i \propto n_i^{\alpha}$, where $\alpha \in (1, 2)$. Therefore, for a given graph, one can learn the best fit for α, and then report the deviations from this model as the outlier scores.

- **Weight feature versus edge feature:** If edges are received unusually frequently in the neighborhood of a node, this will result in a high value of the weight feature relative to the edge feature. This corresponds to the *heavy vicinity* of a node. The expected relationship between these features is $w_i \propto e_i^{\beta}$, where $\beta \geq 1$, but usually ranged in $(1, 1.29)$ according to the experiments in [41].

- **Spectral Feature versus Weight Feature:** This approach detects a single dominating heavy edge in the neighborhood of a node. The expected relationship between these features is $\lambda_i \propto w_i^{\gamma}$, where $\gamma \in (0.5, 1)$.

It should be noted that many features can be mined from the neighborhood of a graph, each of which provides a different notion of an outlier. The aforementioned possibilities simply reflect three *instantiations* of features extracted from a graph neighborhood, so that deviations from power-law models can be used to define outliers. Power laws implicitly encode *domain knowledge* about the behavior of *typical* networks. As discussed frequently in the book, the use of domain knowledge is often very useful for meaningful outlier analysis.

On the other hand, not all features extracted may satisfy such power laws. In general, a wide variety of features can be extracted from the neighborhood of nodes. This can be used to define a multi-dimensional feature space, in which the features extracted from each node correspond to a multi-dimensional data point. For example, the shape of the degree distribution of the neighbors of a node provides a different characterization of the outlier behavior. A general meta-algorithm for node outlier analysis in networks is as follows:

1. Extract features $f_1 \dots f_r$ from the k-hop neighborhood of a node. Create a new multi-dimensional data point $(f_1 \dots f_r)$ specific to that node.

2. Apply any point anomaly detection algorithm to the extracted multi-dimensional data set, and report unusual data points as outliers.

Virtually an unlimited number of different features could be extracted from a node. The precise features extracted should depend on the application at hand. For example, in a social network, it could be a vector containing the number of messages exchanged with each neighboring node, and in a spam detection application, it could correspond to the degree of the node. Therefore, the key in effective node outlier detection is to extract these features in a way that is sensitive to the particular application at hand. In many cases, this process requires a deeper semantic understanding of the significance of the extracted features in that particular graph domain.

12.3.1.1 Leveraging the Mahalanobis Method

The feature extraction methods of the previous section are very effective in cases in which domain knowledge is available about *relevant* characteristics for outlier detection. However, when such domain knowledge is not available, one can use off-the-shelf methods like the Mahalanobis method of section 3.3.8.3. The basic idea is to use the following steps:

1. For a graph G with n nodes create an $n \times n$ similarity matrix S. The adjacency matrix is itself a kind of similarity matrix. However, one can make the similarity matrix more robust by using a number of methods such as the SimRank between pairs of nodes, the Katz measure, the Jaccard coefficient between their neighbor sets, or the personalized PageRank values between pairs of nodes. Note that many of these techniques are used for the link prediction problem in graphs, and are discussed in [33] in detail.

2. Apply the Mahalanobis method of section 3.3.8.3 on similarity matrix S in order to discover anomalous nodes in the graph. Although the Mahalanobis method requires positive semi-definite similarity matrices, section 3.3.8.3 also discusses methods to adjust similarity matrices that are not positive semi-definite.

One can view this approach as a variant of matrix factorization methods, which will be discussed later in this chapter.

12.3.2 Linkage Outliers

Linkage outliers are defined as unexpected edges in particular regions in the network. In most cases, linkage outliers are edges that lie across dense partitions in the network. However, there are numerous models that one can use to define such unexpected edges. The main issue here is to ensure robustness and low computational complexity. The most promising class of algorithms in this category is that of matrix factorization, which is robust and can also be made to be computationally efficient in most settings. The matrix factorization methods can be used to identify both linkage outliers and node outliers. In fact, the Mahalanobis method of the previous section is a special case of this class of techniques in which the factorized basis vectors are mutually orthogonal.

12.3.2.1 Matrix Factorization Methods

Matrix factorization provides a natural way to determine anomalies in data which is represented in the form of a 2-dimensional matrix. Consider a graph $G = (N, A)$ with node set N and an adjacency matrix A. Assume that the number of nodes in N is denoted by n. The adjacency matrix is therefore of size $n \times n$. It is possible for the entries of the adjacency matrix to either be binary or reflect weights on the edges. In most practical settings, the matrix A is very sparse and most of the entries take on zero values. Therefore, the problem of finding linkage anomalies in such graphs can be stated as the problem of determining anomalous entries in a matrix of nonnegative values.

Problem 12.3.1 *Given an $n \times n$ adjacency matrix A, determine anomalous entries in the matrix.*

Matrix factorization provides a natural technique for finding linkage anomalies in such networks. Section 3.5 of Chapter 3 discussed how one can find anomalous entries in a multidimensional data set with the use of matrix factorization. After all, a multidimensional data set can also be represented as a matrix. This general principle applies to *any type of matrix*,

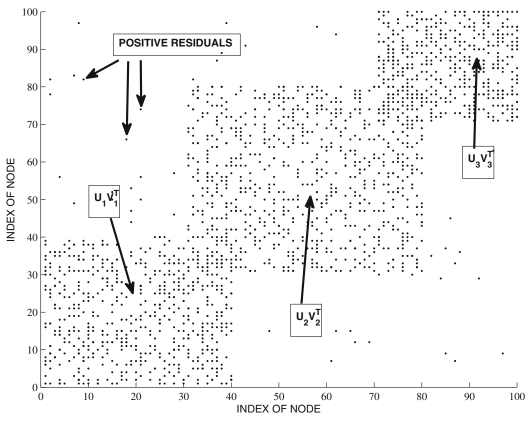

Figure 12.1: Sum-of-parts interpretability of NMF. The matrix has a natural clustering structure each of which is captured by a dominant rank-1 component $U_iV_i^T$. Outliers are not captured by any component and therefore remain as (positive) residuals. Missing entries within a block are negative residuals and are useful for link prediction.

such as an adjacency matrix, and not just matrices constructed from multidimensional data. In fact, in the case of adjacency matrices, the sparsity and nonnegativity can be leveraged to discover more interpretable decompositions in an efficient way. Just as Equation 3.28 of Chapter 3 factorizes a data matrix D, we can decompose the $n \times n$ adjacency matrix A as follows:

$$A \approx UV^T \tag{12.1}$$

Here, U and V are $n \times k$ matrices, where k is the rank of the factorization. The diagonal entries of A are set[1] to the node degrees. In the weighted case, the node degrees are defined as the sum of the entries in each row. One can construct an objective function that minimizes the Frobenius norm of $A - UV^T$ as follows:

$$\begin{aligned} \text{Minimize } & ||A - UV^T||^2 \\ & \text{subject to:} \\ & U \geq 0, \quad V \geq 0 \end{aligned}$$

Although the nonnegativity constraints are not essential, they have some interpretability advantages. Any off-the-shelf optimization solver can be used for this problem, although we will discuss a more efficient approach later in section 12.3.2.1.

This approach can be viewed as a low-rank factorization of A, in which the *residual matrix* $(A - UV^T)$ represents entries that do not conform to the low-rank assumption. Therefore, the low-rank assumption defines the model of normal data. At smaller values of k, the absolute values of the entries in the residual matrix will be larger. Therefore, the

[1]This ensures that the matrix is positive semi-definite and it helps in certain forms of *symmetric* factorization of the form $A = UU^T$, which requires positive semi-definite (PSD) matrices. This is because the adjacency matrix A can now be expressed $A = \sqrt{W}\sqrt{W}^T$, where W is the $n \times m$ *node-edge* incidence matrix. Such a matrix is always PSD. Strictly speaking, the PSD property is not necessary for the type of asymmetric factorization discussed above.

outlier scores are defined by the following residual matrix R:

$$R = A - UV^T \tag{12.2}$$

Clearly, large *absolute* values of the residual correspond to linkage anomalies. Any form of extreme-value analysis may be used on the matrix R to determine the linkage anomalies. Positive values of the residuals correspond to existing edges that should not be present in the graph. Negative values of the residual correspond to edges that should be present, but are not present in the graph. Therefore, they represent the anomaly of *missing edges.* Such methods are, therefore, used for *link prediction* in networks [34], but the absence of such links can also be considered anomalous when the absolute value of the residual is large enough.

Interpretability of NMF Decomposition

Although one can use any form of low-rank factorization, NMF methods have the advantage of being interpretable, and are intimately connected with clustering. Incidentally, clustering is one of the other techniques discussed later in this section for detecting linkage anomalies. Let U_i and V_i be the ith columns of the matrices U and V, respectively. Then, the aforementioned factorization can be expressed in the following additive form:

$$A \approx UV^T = \sum_{i=1}^{k} U_i V_i^T \tag{12.3}$$

Each $U_i V_i^T$ is also nonnegative (because of nonnegativity of U and V), and the positive edges in $U_i V_i^T$ typically represent the edges in a particular community (together with other predicted edges in that community). Note that $U_i V_i^T$ is a rank-1 matrix with a complete block structure. Therefore, the overall decomposition can be viewed as an additive sum of the contributions from k different communities (blocks) in the graph.

In order to understand this point, consider a graph with 100 nodes and three natural clusters, some of which are overlapping. Assume for convenience that the nodes are indexed in order of their clustering structure. In other words, nodes 1 through 40 form one community, the nodes 30 through 80 form another (overlapping) community, and the nodes 70 through 100 form the third community. The nonzero entries in such a 100×100 matrix are shown in Figure 12.1, and they form a natural block structure. Note that some scattered entries in this matrix structure are the residuals, and these entries correspond to the outlier edges. Consider a setting in which we use rank-3 NMF on resulting matrix. Each $U_i V_i^T$ will yield one of the blocks (clusters) in the aforementioned matrix. The scattered entries (outliers) will not be captured by any of the blocks, and will therefore show up as large (positive) residuals.

A different type of matrix factorization is one in which the residuals are constrained to be non-negative for entries in the matrix containing edges [551]. This type of factorization provides different types of insights into the outliers.

Efficiency and Practical Issues for Matrix Factorization

There are several efficiency and practical issues for performing nonnegative matrix factorization. In order to avoid overfitting, it is useful to leverage regularization. Regularization methods are discussed in section 3.5.1 of Chapter 3, in which we add penalties $\alpha(||U||^2 + ||V||^2)$ to the objective function. A second issue is that the underlying matrix may be very large.

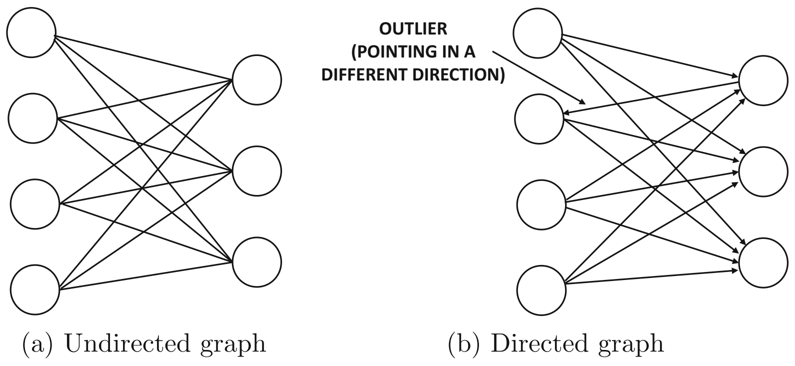

Figure 12.2: Finding outliers in directed graphs. The direction of the edge matters and not just the underlying community structure. The matrix factorization family is general enough to address directed graphs.

For example, for a very modest network containing 10^5 nodes, the number of entries in the matrix is 10^{10}, which creates challenges. Fortunately, such matrices are sparse, and therefore, one can often perform the factorization more efficiently. One interesting approach is to use the sampling of the zero entries. The basic idea is to use all the nonzero entries and to sample the zero entries to perform the factorization using the incomplete matrix factorization approach of section 3.5.1. Some changes need to be made to the gradient-descent steps to accommodate the nonnegativity of factors. To achieve this goal, any negative entries of matrices U and V must always be set to 0 after each gradient-descent step. Furthermore, one must always initialize the entries in U and V to nonnegative values. The resulting factorization can be used to create an outlier score of each non-diagonal entry in the adjacency matrix. This process can be repeated multiple times, and the scores can be averaged to provide a final prediction.

In addition to estimating the outlier scores of links, one can also score nodes. The outlier score of node i is obtained by averaging the squared (edge) scores in both row i and column i in R. The scores of all non-diagonal entries in the ith row and column are used, whether or not they correspond to links that are actually present in the graph.

Application to Directed Graphs

One of the nice aspects of the matrix factorization methodology is that it can be easily applied to directed graphs, and it is not restricted to undirected graphs. For example, the undirected graph in Figure 12.2(a) is a complete bipartite graph, and it has no outlier edge. However, the graph in Figure 12.2(b) is derived from Figure 12.2(a) by adding directions to the edges. In this case, there is a single outlier edge in Figure 12.2(b), which is caused primarily by its direction. Such outliers can also be identified by using the aforementioned matrix factorization methodology. The only difference is that an asymmetric adjacency matrix is factorized in this case, and the matrices U and V will be interpreted as *sender* and *receiver* factors. The steps for identifying outliers are otherwise identical to the undirected case.

12.3.2.2 Spectral Methods and Embeddings

The matrix factorization methodology discussed above can be (conceptually) replicated with the use of spectral methods, at least for the case of undirected graphs. In general, matrix factorization methods are known to be roughly equivalent to spectral methods. Note that we can factorize the adjacency matrix A directly into orthogonal factors using eigen-decomposition. This is (almost) equivalent to using (unnormalized) spectral methods. Note that A is positive semi-definite with nonnegative eigenvalues when the diagonal entries of A are set to be equal to the node degrees. Therefore, one can factorize A as follows:

$$A \approx P\Lambda P^T = (P\sqrt{\Lambda})(P\sqrt{\Lambda})^T \tag{12.4}$$

Here, P is an $n \times k$ matrix, whose columns contain the orthonormal eigenvectors (with largest eigenvalues), and Λ is a diagonal $k \times k$ matrix containing the nonnegative eigenvalues. One can also view this approach as a type of symmetric matrix factorization $A \approx UV^T$, in which both U and V are set to $P\sqrt{\Lambda}$. Furthermore, instead of nonnegativity, the columns of U and V satisfy the constraint that they are mutually orthogonal. Therefore, this spectral approach also falls squarely into the category of matrix factorization methods. As before, the residual matrix may be computed as follows:

$$R = A - UV^T = A - (P\sqrt{\Lambda})(P\sqrt{\Lambda})^T \tag{12.5}$$

Nondiagonal entries with large (positive) residual values may be declared anomalies. These methods are intuitively very similar to the matrix factorization methods discussed above, except that PCA is used for determining the residuals. Of course, the actual values of the residuals will be somewhat different, since the models are not mathematically identical. The interpretability of such methods is lower, since it is no longer possible to impose non-negativity constraints on the factors. The matrix $(P\sqrt{\Lambda})$ is an embedding of the nodes in multidimensional space. However, this type of symmetric factorization has another advantage in terms of using the embedding directly in conjunction with multidimensional *node* outlier detection methods. In fact, the Mahalanobis method of section 12.3.1.1 is an example of such an approach.

Leveraging the Embedding

Even though matrix factorization methods are designed for discovering edge anomalies, the resulting embeddings can also be leveraged for discovering node anomalies. Note that $P\sqrt{\Lambda}$ provides a k-dimensional embedding of the nodes. In order to discover the outliers, we set k to the smallest value, such that all[2] nonzero eigenvectors are captured in P. The matrix P provides a normalized embedding, in which each of the k dimensions has unit variance. Subsequently, the Euclidean distance of each row to the mean of the rows of P provides the outlier score of that node. This approach is equivalent to the use of the kernelized PCA version of the Mahalanobis method (see Chapter 3). Furthermore, one can use many other multidimensional outlier detection techniques on this representation.

Note that one can use other kernel methods such as diffusion kernels, or use other types of node-node similarity matrices (such as the Katz, SimRank, or personalized PageRank measures), to perform the factorization and generate the embedding. These different types of embeddings will work well in different application domains. In many cases, it is useful to

[2]One must be careful not to be misled by nonzero eigenvalues caused by numerical errors. Such eigenvalues are usually obvious because of their extremely small values. As a practical matter, one can set a very small threshold, such as 10^{-8}.

have some insight about a relevant and application-specific notion of similarity to generate an effective embedding.

12.3.2.3 Clustering Methods

Nonnegative matrix factorization is only one of the many ways in which one might cluster a network to determine outlier edges. In general, if the nodes in the graph are clustered, then the edges across these node clusters are defined as outliers. Unfortunately, there is no single way of defining the clusters in a network. In many cases, the clustering may be imperfect, and may merge two sets of nodes that should ideally be in different clusters. The source of this problem is that graphs often contain overlapping clusters, and a strict partitioning may be defined in many ways by a randomized algorithm. In such cases, different clusterings might result in different edges being reported as outliers. One approach for increasing robustness is to use randomized ensembles.

Randomized Ensembles

The method in [17] defines a clustering that is based on randomized sampling of the edges in the network. This is a special case of subsampling-based ensembles for outlier detection (cf. section 6.4.6.2 of Chapter 6), although the sampling is performed on the *entries* (edges) of the adjacency matrix rather than its *rows*. A sample of edges from a stream implicitly creates a set of clusters in terms of the connected components in this sample. Such connected components are much denser than randomly selected node sets in the graph, because of the inherent bias of edge sampling. Since the connected components in edge samples are known to create *weak clusters* of densely connected nodes, such an approach is an excellent strategy for link outlier estimation, when used within the context of an ensemble framework. In other words, this randomized sampling is repeated in order to create the clusters in different ways. For each clustering, a likelihood fit is defined for an edge. The median of these likelihood fits is defined as the final outlier score. Furthermore, the approach in [17] modifies the edge sampling in order to maintain specific properties of node partitions such as balancing the number of nodes in each cluster (see section 12.5.2).

Consider a partitioning of the nodes denoted by $\mathcal{C} = C_1 \dots C_k$. The number of node partitions in $\mathcal{C}$ is denoted by k. Each set C_i represents a disjoint subset of the nodes in V. The likelihood fit is defined with respect to a *structural generation model* of edges with respect to node partitioning $\mathcal{C}$. This defines the probability of an edge between a pair of partitions and it also provides an outlier score for the edge. Lower values indicate that a data point is an outlier.

Definition 12.3.1 (Edge Generation Model [17]) *The structural generation model of a node partitioning $\mathcal{C} = \{C_1 \dots C_k\}$ is defined as a set of k^2 probabilities $p_{ij}(\mathcal{C})$, such that $p_{ij}(\mathcal{C})$ is the probability of a randomly chosen edge in the network to be incident on partitions i and j.*

The probabilities $p_{ij}(\mathcal{C})$ are estimated from the fraction of edges between various pairs of clusters. The partition identifier for node i is denoted by $I(i, \mathcal{C})$, and it takes on a value between 1 and k.

Definition 12.3.2 (Edge Likelihood Fit) *Consider an edge (i, j), a node partition $\mathcal{C}$, and edge generation probabilities $p_{ij}(\mathcal{C})$ with respect to the partition. Then, the likelihood* *fit of the edge (i, j) with respect to the partition $\mathcal{C}$ is denoted by $\mathcal{F}(i, j, \mathcal{C})$ and is given by $p_{I(i,\mathcal{C}),I(j,\mathcal{C})}$.*

The likelihood fit of an edge can be defined with respect to any partition in the network. It remains to be explained how the likelihood fits from different partitions can be combined.

The r different ways of creating the partitions are denoted by $\mathcal{C}_1 \dots \mathcal{C}_r$. Specifically, the ith clustering $\mathcal{C}_i$ contains $k(\mathcal{C}_i)$ different clusters. The composite edge-likelihood fit from these r different ways of creating the partitions is defined as the *median* of the edge likelihood fits over the different partitions.

Definition 12.3.3 (Edge Likelihood Fit (Composite)) *The composite edge likelihood fit over the different clusterings $\mathcal{C}_1 \dots \mathcal{C}_r$ for the edge (i, j) is the median of the values of $\mathcal{F}(i, j, \mathcal{C}_1) \dots \mathcal{F}(i, j, \mathcal{C}_r)$. This value is denoted by $\mathcal{MF}(i, j, \mathcal{C}_1 \dots \mathcal{C}_r)$.*

A variety of randomized clustering methods can be used to create the different partitions. For example, any off-the-shelf randomized algorithm can be used for partitioning purposes. However, the edge sampling technique has the virtue of being efficient to implement, and it is also helpful in performing effective variance reduction. A method has also been proposed [17] to maintain the clusterings for a *stream of edges* with an online reservoir sampling approach. Thus, this approach can be used to determine edge anomalies in graph streams. The reservoir sampling approach will be discussed in detail in section 12.5.2.

12.3.2.4 Community Linkage Outliers

In some network applications, many small graphs may be superimposed on a single large graph. For example, in a bibliographic network, a small graph object may represent a publication. In such cases, it is desirable to identify objects, for which the linkage structure is anomalous compared to the linkage structure of other objects in the network. This case is often not very different from determining linkage outliers. The main difference is that the likelihood scores for the different edges in the object need to combined into a single likelihood score [17].

The likelihood fit for a graph object G is the product of the likelihood fits of the edges in G. In order to fairly compare between graphs which contain different numbers of edges, we put the fraction $1/|G|$ in the exponent, where $|G|$ is the number of edges in the incoming graph stream object. In other words, we use the geometric mean of the likelihood fits of different edges in the incoming graph stream object. Therefore, we define the object likelihood fit as follows.

Definition 12.3.4 (Graph Object Likelihood Fit) *The likelihood fit $\mathcal{GF}(G, \mathcal{C}_1 \dots \mathcal{C}_r)$ for a graph object G with respect to the partitions $\mathcal{C}_1 \dots \mathcal{C}_r$ is the geometric mean of the (composite) likelihood fits of the edges in G.*

$$\mathcal{GF}(G, \mathcal{C}_1 \dots \mathcal{C}_r) = \left[\prod_{(i,j) \in G} \mathcal{MF}(i, j, \mathcal{C}_1 \dots \mathcal{C}_r) \right]^{1/|G|} \tag{12.6}$$

Objects that have extremely low likelihood fits are reported as community-linkage outliers [17]. In many cases, any particular linkage in the set may not have a low fit, but the overall fit, when evaluated over the entire set, may be very low. Thus, such outliers are examples of *collective* outliers, whereas node and linkage outliers are *contextual* outliers.

Strictly speaking, this measure can be defined for any set of nodes in a single large network, rather than a specified set of nodes in the small graph superimposed over a larger network. In such cases, an additional challenge is to *discover* the appropriate sets of nodes that are connected together in an anomalous way. This is a much more difficult problem with the use of only network structure information, unless additional content-based supervision is available. A closely related notion of community outliers was proposed in [214], which finds such anomalous communities in a single large network. However, node content is used to provide additional supervision for the discovery of such anomalous sets of nodes.

12.3.3 Subgraph Outliers

A subgraph outlier is defined as a part of the graph which exhibits unusual behavior with respect to the normal patterns in the full graph. Subgraph outliers cannot be defined meaningfully in a large graph, unless there are repetitions in the labels on the nodes. For example, in a Web or social network graph, in which all node identifiers are distinct, it is much harder to define a notion of regularity. However, if the nodes are associated with labels drawn from a relatively small subset of possibilities, it is much easier to define subgraph outliers by finding subgraphs which relate these labels to one another in unusual ways. Alternatively, the nodes may not have labels associated with them, and the matching is based purely on structural similarity. Either of these scenarios can be addressed by the work in [416], which designs information-theoretic models for subgraph outlier detection.

The approach proposed in [416] is based on the Minimum Description Length (MDL) principle for outlier detection. The broader principles underlying the approach are based on the SUBDUE system for subgraph pattern mining. In the subdue system, a commonly occurring pattern is one which provides a small description length for the purposes of representation. For example, if S is a substructure which occurs repeatedly in a graph G, then by compressing that substructure into a single vertex, the graph is compressed (corresponding to a smaller description length), and the overall graph can be represented in terms of the description of the compressed graph and the smaller substructure. Therefore, the frequent nature of a substructure S in graph G may be leveraged to describe the compressed representation of the graph concisely in terms of the description length $DL(G|S)$ of the compressed graph and that of the repeatedly occurring substructure S. The conciseness is achieved, because the repeatedly occurring substructure needs to be described only once. This description length $F1(S, G)$ is defined as follows:

$$F1(S, G) = DL(G|S) + DL(S) \tag{12.7}$$

Subgraphs that are very common (and therefore not outliers) will have low values of $F1(S, G)$. One possible solution is to identify those subgraphs that have *high values* of $F1(S, G)$ and flag them as outliers. Such an approach does not seem to work very well for anomaly detection because it does not discriminate between the varying sizes of subgraphs. For example, subgraphs with only one node will often have high values of the outlier score according to this criterion. Therefore, a different heuristic criterion was proposed in [416], which is based on the spirit of the original definition for pattern frequency, but normalizes for subgraph size as well. Specifically, this definition is as follows:

$$F2(S, G) = Size(S) * Instances(S, G) \tag{12.8}$$

Here, $Size(S)$ represents the number of nodes in the subgraph S, and $Instances(S, G)$ represents the number of instances of subgraph S in the graph G. Subgraphs with low value of $F2(S, G)$ are reported as outliers.

A second approach is based more directly on the SUBDUE method [151]. The SUBDUE method is designed to determine the common substructures in the graph in an iterative way. Specifically, this method compresses the common substructures in the graphs, and replaces them with new nodes. Substructures that are more common are compressed in earlier iterations. The key insight in detecting anomalous subgraphs is in determining whether significant portions of the subgraph were compressed in early iterations. If this is *not* the case, it implies that the subgraph contains few common substructures. Such a subgraph should be considered anomalous. Therefore, the outlier score O of a subgraph S is defined in terms of the iteration index i of the SUBDUE method as follows:

$$O = 1 - \sum_{i=1}^{n} \frac{(n-i+1)}{n} \cdot \frac{DL_{i-1}(S) - DL_i(S)}{DL_0(S)} \tag{12.9}$$

The outlier score always lies in the range $(0, 1)$. High values indicate that the object is anomalous. The description length $DL_i(S)$ of a subgraph typically reduces with increasing iteration index i, because of the compression of nodes inside it. Note that the term $\frac{DL_{i-1}(S)-DL_i(S)}{DL_0(S)}$ corresponds to the fraction of compression in the ith iteration, and the term $\frac{(n-i+1)}{n}$ provides greater weight to a compression occurring in an earlier iteration. This is because more common graphs are compressed in earlier iterations. Therefore, the value of the outlier score will be impacted by high quickly the nodes of the subgraph S become compressed because of the membership of common patterns in them. This provides a heuristic definition of the outlier score, based on the MDL principle.

12.4 Node Content in Outlier Analysis

The incorporation of node content can have significant benefits in outlier analysis, particularly in terms of discovery of outlier links between nodes. In most application domains such as the Web, social networks, and information networks, linkages between nodes are highly correlated to content similarity between nodes. For example, Web pages with similar content are more likely to be linked. In social networks, network participants are more likely to be linked to other participants with similar interests. On the other hand, nodes with certain types of content are very unlikely to link with one another. For example, an official US Government Web site is unlikely to link to Web sites encouraging criminal endeavors for the most part. Discovering outlier linkages in such settings can have numerous applications:

- Noisy outlier linkages can degrade the accuracy of network mining algorithms. Therefore, it may be useful to detect and remove such linkages before performing the mining. This problem is referred to as *network denoising* [212, 452].

- Anomalous linkages can provide insights about unusual connections among nodes. This may have applications beyond noise removal, since the formation of anomalous connections is indicative of interesting facts about the underlying system.

These methods often use consistency modeling between the content and clustering structure to identify linkage anomalies.

12.4.1 Shared Matrix Factorization

The matrix factorization methods discussed in the previous section can also be extended to include content. The key idea is to use *shared* matrix factorization. Aside from the $n \times n$

adjacency matrix A, we also have an $n \times d$ content matrix C, where d is the size of the underlying lexicon. Then, the idea is to simultaneously factorize A and C as follows:

$$A \approx UV^T \tag{12.10}$$

$$C \approx UW^T \tag{12.11}$$

Here, U and V are $n \times k$ matrices, whereas W is a $d \times k$ matrix. Note that the matrix U is *shared* in both factorizations. Then, one can determine the appropriate factors by using the following optimization formulation:

$$\begin{aligned} \text{Minimize } & ||A - UV^T||^2 + \beta||C - UW^T||^2 + \alpha(||U||^2 + ||V||^2 + ||W||^2) \\ & \text{subject to:} \\ & U, V, W \geq 0 \end{aligned}$$

Here, $\beta > 0$ is the balancing parameter and $\alpha > 0$ is the regularization parameter. One can initialize the matrices with nonnegative entries and use gradient-descent methods to solve this problem. Let $E_A = (A - UV^T)$ and $E_C = (C - UW^T)$ be the error matrices at any particular step of the descent. Then, the gradient descent steps are as follows (with step-size $\eta > 0$):

$$\begin{aligned} U &\Leftarrow U(1 - \eta\alpha) + \eta E_A V + \eta\beta E_C W \\ V &\Leftarrow V(1 - \eta\alpha) + \eta E_A^T U \\ W &\Leftarrow W(1 - \eta\alpha) + \eta\beta E_C^T U \end{aligned}$$

At each step, the entries in U, V, and W that become negative are set to 0 in order to respect the nonnegativity constraint. The nonnegativity constraint provides better interpretability, but it is not essential to the basic formulation. Interested readers should refer[3] to [34] for more details on the derivation.

After the optimal solutions for U, V, and W have been determined, the anomalous edges in A can be determined from the large positive residuals of $(A - UV^T)$. Similarly, the anomalous words in C can be determined from the large positive residuals in $(C - UV^T)$. Large negative residuals correspond to missing edges or missing words. Anomalous nodes can also be determined by summing up the squares of the residuals of the relevant residuals in A and C. For example, for the ith node, we can sum up the squares of the residuals of the ith rows in A and C to evaluate the outlier score of node i.

12.4.2 Relating Feature Similarity to Tie Strength

Another recent method [212] models the similarity in the feature vectors at the nodes to an intuitive notion of tie-strength across links. This tie-strength is modeled as a weight vector, and represents the consistency of links with their neighborhood feature vector. Each node i has a k-dimensional (column) feature vector $\overline{f_i}$ associated with it. Let F_i be a $k \times d_i$ matrix of content weights (for the neighborhood of node i) with rows representing the k features and columns representing its d_i neighbors. Let $\overline{w_i}$ be a non-negative column vector of length d_i containing the tie-strength of node i to each of its d_i neighbors. Then, $F_i\overline{w_i}$ represents the set of predicted features at node i based on the structure and content of the

[3]A solution to a very similar optimization formulation is provided in section 11.3.8 of Chapter 11 of [34]. However, this solution is provided in the completely different context of recommender systems with trust matrices.

neighborhood of node i. The error of this prediction is given by $||F_i\overline{w_i} - \overline{f_i}||_2^2$. In addition, an L_1-regularization penalty is imposed on the weight vector to encourage sparsity. The following objective function is minimized:

$$\text{Minimize}_{\overline{w_i} \geq 0} \sum_i (||F_i\overline{w_i} - \overline{f_i}||_2^2 + \lambda \cdot ||\overline{w_i}||_1) \qquad (12.12)$$

Here, $\lambda > 0$ is a regularization parameter that regulates the tradeoff between the sparsity of the weight vector and the learned accuracy. Links that correspond to very small values of the weights (or zero weights) are assumed to be outliers. Larger values of λ will result in sparser weight vectors, and, therefore, fewer links will be included. This is a convex non-linear programming problem for which many optimization solvers are available [112].

12.4.3 Heterogeneous Markov Random Fields

A recent method [452] uses a heterogeneous Markov random field framework (HMRF) to evaluate the consistency of links with the clustering structure of the network. Therefore, this model uses a binary variable $n(i, j)$ to quantify whether or not a link should be considered anomalous. Similarly, a node-specific variable indicates its affinity to each cluster. Then, the two variables are *simultaneously* learned to identify the clusters and outlier links in the network. The criteria for learning these variables leverage the consistency between network clustering structure and node content. The key idea is to create clusters and infer the anomalousness of links simultaneously in a *mutually consistent way.* At the same time, the linkages that are inconsistent with the learned variables are deemed to be anomalous. A detailed introduction of HMRF models is beyond the scope of this book. Interested readers are referred to [452] for details.

Finally, the *community-outlier detection* method [214] identifies outlier communities by using a combination of content-based and structural analysis. Linkage outliers try to find *a pair* of unusually connected nodes in the network. Community-outliers can be considered unusual subgraphs in the network and are therefore a generalization of linkage outliers. The technique in [214] proposes a method for efficient identification of community outliers with a heterogeneous Markov random field (HMRF) model. Interested readers are referred to [214] for details.

12.5 Change-Based Outliers in Temporal Graphs

Many entities such as the Web, social, and information networks are temporal in nature. This has led to increasing interest in the field of *evolutionary network analysis* [14]. In such cases, unusual and abrupt changes in the structure of the network should be detected and reported as anomalies. As discussed in earlier chapters, outlier detection and change analysis are tightly integrated problems [193, 579], especially when the changes are abrupt and reflect unusual events. In the context of graphs and networks, one can characterize these changes in various ways:

- **Node anomalies:** Sudden changes in the structural patterns in the vicinity of a node can lead to node anomalies. Such anomalies are also referred to as *hotspots* [597]. For example, if a researcher suddenly starts publishing with a new set of co-authors in a bibliographic network, it will lead to an anomaly.

- **Linkage anomalies:** New edges that are inconsistent with the current graph structure may be considered anomalous. For example, the arrival of a link that bridges two sparsely connected regions of the network may be considered anomalous. This is similar to the static case, except that only the past history of the stream is relevant for modeling.
- **Community evolution:** In this case, significant and unusual changes in the community structure are tracked [20, 233, 519] and reported as anomalies.
- **Distance evolution:** In this case, pairs of nodes with unusually large changes in the distances are detected [234] and reported as anomalies.
- **Latent embedding methods:** These are very general methods that discover temporal embeddings from the sequence of graph snapshots. A temporal smoothness constraint is incorporated in the embedding. Violations of this constraint represent change points.

In certain types of networks such as communication networks, the edges may be received as a fast stream. The streaming scenario poses special challenges because it is important to be able to build the model of normal data and report outliers in an online setting.

12.5.1 Discovering Node Hotspots in Graph Streams

Unusual activity in a network might cause unusual changes in the neighboring network structure. These unusual changes are also referred to as *hotspots*. In order to discover such hotspots, the work in [597] performs *localized* eigenvector analysis. The idea is to create a matrix characterizing the immediate locality of a particular node. This matrix is created as a decay-based version of the adjacency matrix of the immediate locality of the node. Specifically, if $A(i)$ is the adjacency matrix directly adjacent to the node i, then the matrix $M(i) = A(i)^T A(i)$ is used to characterize the neighborhood of node i. This matrix is maintained in conjunction with a decay-function in online fashion. It has been shown in [597] how such a matrix can be maintained efficiently with the use of lazy updates.

In order to determine anomalies, the eigenvectors of $M(i)$ are determined. Note that the eigenvectors of $M(i)$ are the same as the right singular vectors of $A(i)$. Therefore, this approach is essentially a matrix factorization method. The magnitude and direction of the dominant eigenvector of this matrix will change over time as the local structure near the node changes over time. Furthermore, abrupt changes in these eigenvectors can also be helpful in discovering anomalous changes. There are two primary types of anomalous changes:

1. If the magnitude of the eigenvector changes abruptly, it means that there is a significant change in the level of activity near that node. For example, in a bibliographic network, a sudden increase in publication volume of an individual will increase the magnitude of the eigenvector.
2. If the angle of the eigenvector changes abruptly, it means that the pattern of linkages in the locality of the node has changed abruptly. In a bibliographic network, a sudden change in the pattern of co-authors will lead to a change in the direction of the eigenvector.

The level of change in both cases can be reported continuously as an outlier score. One can use a threshold on this outlier score in order to trigger alarms about significant events.

It is noteworthy that PCA is a form of matrix factorization. It is possible to use this approach in conjunction with other forms of matrix factorization. Other forms of factorization such as NMF might provide better interpretability in many settings.

12.5.2 Streaming Detection of Linkage Anomalies

The linkage anomaly detection method in section 12.3.2 can be extended to graph streams [17]. The only difference is that the likelihood estimation for a linkage anomaly needs to be computed with respect to the previous history of the graph stream. In this case, cluster-based partitions need to be maintained dynamically from the edge stream[4] to perform the linkage anomaly detection. It is well-known that the use of edge sampling can be used to create dense partitions. The main problem is that specific *structural properties* of the k-way cut partitions need to be maintained. For example, one might want to ensure a minimum number of points in each cluster, or to constrain the total number of clusters. Clearly, a random edge sample may not satisfy such constraints. This goal is achieved with the help of a *monotonic set function* of the underlying edges in the reservoir. A monotonic set function is defined on the sample as follows.

Definition 12.5.1 (Monotonic Set Function) *A monotonically non-decreasing (non-increasing) set function is a function $f(\cdot)$ whose argument is a set, and value is a real number which always satisfies the following property:*

- *If S_1 is a superset (subset) of S_2, then $f(S_1) \geq f(S_2)$*

Some examples of monotonic set functions are as follows:

- The function value is the number of connected components in the edge set S (monotonically non-increasing).

- The function value is the number of nodes in the largest connected component in edge set S (monotonically non-decreasing).

In order to maintain the desired structural properties on the induced clustering (e.g., cluster balance), it is possible to use *thresholds* on the above properties, which are also referred to as *stochastic stopping criteria*. Let $\mathcal{D}$ be a set of edges. Consider a restricted bunch of subsets of $\mathcal{D}$ in which the edges are sorted and can be added to S only in *sort order priority*; in other words, an edge cannot be included in a subset S, if all elements occurring before it in the sort order are also included. Clearly, the number of such subsets of $\mathcal{D}$ is linear in the size of $\mathcal{D}$.

Definition 12.5.2 (Sort Sample with Stopping Criterion) *Let $\mathcal{D}$ be a set of edges. Let $f(\cdot)$ be a monotonically non-decreasing (non-increasing) set function defined on the edges. A sort sample S from $\mathcal{D}$ with stopping threshold α is defined as follows:*

- *All edges in $\mathcal{D}$ are sorted in random order.*

- *The* smallest *subset S from $\mathcal{D}$* among all subsets satisfying the sort-order priority *is identified, such that $f(S)$ is at least (at most) α.*

[4]The work in [17] uses a stream of composite objects, although this presentation simplifies it to edge streams. Either scenario can be handled by the approach with small modifications.

This means that if the last added element is removed, then that set (and all previous subsets) will not satisfy the stopping criterion. As a practical matter, the set obtained by removing the last added element is the most useful for processing purposes. For example, if $f(S)$ is the size of the largest connected component, the stopping criterion determines the smallest sample S in which the size of the largest connected component is at least a user-defined threshold α. By dropping the last edge (v, w) that was added to S, it is guaranteed that the size of the largest connected component in $S - \{(v, w)\}$ is less than α. Correspondingly, a *penultimate set* for a sort sample is defined as follows.

Definition 12.5.3 (Penultimate Set) *The penultimate set for a sort sample S is obtained by removing the last element in the sort order of sample S.*

For the case of a *fixed data set*, it is fairly easy to create a sort sample with a structural stopping criterion. This is achieved by sorting the edges in random order and adding them sequentially, until the stopping criterion can no longer be satisfied. However, in the case of a data stream, a random sample or reservoir needs to be maintained *dynamically.* Once edges have been dropped from the sample, it becomes a challenge to compare their sort order to incoming edges.

The key idea is use a *fixed random hash function*, which is computed as a function of the node labels on the edge, and remains fixed over the entire stream. This hash function is used to create a sort order among the different edges. This hash function serves to provide *landmarks* for incoming edges when they are compared to the previously received edges from the data stream. Furthermore, the use of a hash function *fixes the sort order among the edges throughout the stream computation.* The fixing of the sort order is critical in being able to design an effective structural sampling algorithm. Therefore, for an edge (i, j) the hash function $h(i \oplus j)$ is computed, where $i \oplus j$ is the concatenation of the node labels i and j. The use of a sort on a random hash function value induces a random sort order on the stream elements. Furthermore, a stopping criterion on the sorted data set *translates* to a threshold on the hash function value. This provides an effective way to control the sampling process. In other words, it is desirable to determine the smallest threshold q, such that the set S of edges that have hash function value at most q satisfy the condition that $f(S)$ is at least (at most) α. The key observation here is that the value of the threshold q varies over the life of the data set, as more edges arrive, and a smaller *fraction* of the edges in the stream need to be included in the reservoir to satisfy the structural constraint. Intuitively, this "smaller" fraction translates to lower hash thresholds. In fact, the following result has been explicitly shown in [17]:

Theorem 12.5.1 (Hash Threshold Monotonicity) *The stopping hash threshold is monotonically non-increasing over the life of the data stream.*

This result implies that edges that have not been included in the current sample will *never* be relevant for sampling over the future life of the data stream. Thus, there is no risk that any stream edge which was discarded will become useful later. The current sample is the only set needed for any future decisions about reservoir sample maintenance. The key here is to find ways to dynamically update the hash threshold and the stream sample continuously so as to maintain the structural constraints.

A simple algorithm is used to dynamically maintain the reservoir. The *current hash threshold* is maintained dynamically to make decisions on whether or not incoming elements are included in the reservoir. For each incoming edge, the hash function is applied, and it is added to the reservoir, if the hash function value is less than the current threshold value. The addition of an edge will always result in the stopping criterion being met because of set function monotonicity. However, the set may no longer be the *smallest sort-ordered set* to do so. Therefore, edges may need to be removed in order to make it the smallest sort-ordered set to satisfy the stopping criterion. In order to achieve this goal, the edges in the reservoir are processed *in decreasing order of the hash function value* and removed, until the resulting reservoir is the smallest possible set which satisfies the stopping constraint. The corresponding hash threshold is then reduced to the largest hash function value of the *remaining edges in the reservoir* after removal. Clearly, the removal of edges may result in a reduction of the hash threshold in each iteration. However, it will never result in an increase in the threshold, because all the added edges had a hash function value lower than the threshold in the previous iteration. The penultimate set derived from the sort sample is always used for the purposes of maintaining the cluster partition as the set of underlying connected components.

The above description explains the maintenance of a single reservoir (and corresponding partition). For robustness, a set of r different reservoirs is maintained, which corresponds to r different partitionings of the data. Therefore, r different hash functions are used to create the different reservoir samples. These are used to define the r different node partitionings required by the outlier modeling algorithm. The *penultimate sets* of the r different reservoirs are denoted by $S_1 \ldots S_r$. These will be used for the purpose of inducing the r different partitionings denoted by $\mathcal{C}_1 \ldots \mathcal{C}_r$. Correspondingly, the outlier score of an edge can be computed from the likelihood fits, as discussed in section 12.3.2 of this chapter.

12.5.3 Outliers Based on Community Evolution

Evolutionary changes in a dynamic network often cause significant changes in the structure of the underlying communities. The analysis of community evolution reveals important insights about sudden changes in membership of nodes, the formation or disappearance of clusters, the erratic membership of nodes in clusters, and a general change in the overall clustering quality. Many of these algorithms integrate clustering maintenance with evolution analysis, but this is not always the case [14].

12.5.3.1 Integrating Clustering Maintenance with Evolution Analysis

While evolutionary clustering [119, 143] is widely studied in the community detection literature, a recent method [233] also integrates the clustering process with evolution analysis. The integration of an evolution analysis procedure into the clustering process is critical in determining relevant change points in the data. The work in [233] uses a probabilistic clustering model to learn the clusters from the underlying data.

The *ENetClus* method [233] generalizes the probabilistic *NetClus* [525] model to the temporal scenario. This is a soft clustering model, which assigns probabilities of membership of each node to different clusters. The idea in this model is to perform the clustering on temporal snapshots of the data. On each snapshot, a probabilistic assignment is learned with the use of the *NetClus* algorithm. The final probabilistic assignment in a given snapshot is used as an initialization point (prior) for the next iteration. This ensures that continuity is maintained among the clusters, and the clusters found in the next snapshot can be directly compared to their counterpart in the current snapshot. A number of evolution metrics are then proposed in order to measure significant clustering properties in the form of a time-series stream. Significant deviations in these values (by using the methods discussed in Chapter 9) can be reported as anomalous changes in the network. A subset of the more important temporal change quantifications introduced in [233] are presented below:

- **Cluster membership consistency:** The vector of probabilities of membership of a node to different clusters is available in soft clustering algorithms like *ENetClus*. The cosine similarity between the vectors in successive snapshots can be reported as the change value.

- **Cluster snapshot quality:** The ratio of intra-cluster similarity to inter-cluster similarity is used as a quantification of the quality of the clusters. Significant changes in the clustering quality between successive snapshots is indicative of the fact that the inherent clustering tendency of the network has changed significantly over time. For example, if a network receives a significant number of spam links in a short period of time (an anomalous event), this could abruptly disrupt the clustering tendency of the data set. This will show up as a sudden change in the time series representing a quantification of the clustering quality.

- **Cluster novelties, merges, splits, and disappearances:** The formation of new clusters represents a novelty in the data. Similarly, significant structural changes may occur, corresponding to cluster merge, split or disappearance. Each of these different structural anomalies is quantified in [233].

- **Temporal object stability:** Objects that move across different clusters over time are inherently unstable and should therefore be considered outliers. The fraction of timestamps at which the membership of the cluster remains the same between successive snapshots is a definition of temporal object stability. The inverse of this quantity is the outlier score for the object.

- **Temporal object sociability:** Objects that inherently belong to many different clusters are considered sociable. In the context of a soft clustering algorithm, this means that the membership probability vector for the object is distributed across different clusters. This definition of sociability is different from the degree of a node, since it is performed in the context of the community behavior, which provides more intuitive insights into aggregate sociability trends. This can be quantified by using the Gini index or the entropy for the object in terms of cluster membership probabilities. Let $p_1 \ldots p_k$ be the membership probabilities for an object in the k different clusters. The Gini index is defined by the expression $1 - \sum_{i=1}^{k} p_i^2$, and the entropy is defined by the expression $-\sum_{i=1}^{k} p_i \cdot \log(p_i)/k$.

  Larger values of each of these quantifications indicate greater sociability. For example, in a bibliographic network, this corresponds to authors who collaborate across many different research areas. Sudden changes in the sociability can provide an understanding of the significant changes in the collaboration patterns of a particular author.

In general, the use of any of these metrics is dependent on the specific type of application at hand.

One can also use other clustering methods, such as spectral methods, for the aforementioned analysis. However, this results in a hard clustering and the aforementioned quantifications would need to be designed differently. A method for incorporating temporal smoothness in spectral clustering algorithms is discussed in [143]. While this method is not designed explicitly for change detection, it can be used for change detection, since an approximate mapping can be found between clusters at different time snapshots. This is because of the incorporation of the temporal smoothness criterion, which allows a clear mapping between clusters at different snapshots. A specific application of this kind of approach to the monitoring of evolution in blog communities is discussed in [415].

12.5.3.2 Online Analysis of Community Evolution in Graph Streams

The work in [20] discovers the use of highly expanding or highly contracting communities with the use of the concept of a *differential graph.* The differential graph over interval (t_1, t_2) is defined as the fractional rate at which the level of interaction along an edge has changed in that period. Let $w_{ij}(t)$ be the weight of edge (i, j) at time t based on its frequency of arrival. In order to generate the differential graph, the *normalized graphs* at times t_1 and t_2 are constructed. The normalized graph $G(t) = (N(t), A(t))$ at time t is denoted by $\overline{G(t)}$, and contains exactly the same node and edge set, but with different weights. Let $W(t) = \sum_{(i,j)\in A} w_{ij}(t)$ be the sum of the weights over all edges in the graph $G(t)$. Then, the normalized weight $\overline{w_{ij}(t)}$ is defined as $w_{ij}(t)/W(t)$. The normalized graph therefore contains the fraction of interactions of each edge. The differential graph is constructed from the normalized graph by using a subtractive process on the normalized graphs at snapshots t_1 and t_2. Therefore, the differential graph $\Delta G(t_1, t_2)$ contains the same nodes and edges as $\overline{G(t_2)}$, except that the differential weight $\Delta w_{ij}(t_1, t_2)$ on the edge (i, j) is defined as $\Delta w_{ij}(t_1, t_2) = \overline{w_{ij}(t_2)} - \overline{w_{ij}(t_1)}$.

In the event that an edge (i, j) does not exist in the graph $\overline{G(t_1)}$, the value of $w_{ij}(t_1)$ is assumed to be 0. Because of the normalization process, the differential weights on many of the edges may be negative. These correspond to edges over which the interaction has reduced significantly during the evolution process. For instance, in a bibliographic network, when the *rate of authoring new publications* between a pair of authors reduces over time, the corresponding weights in the differential graph are also negative.

After the differential graph has been constructed, it is desired to determine highly evolving node subgraphs. Edges with highly positive or negative weights correspond to evolving entities. Therefore, in order to identify expanding and contracting communities, it is desired to identify subgraphs in which most interactions are all either highly positive or highly negative. This is a much more difficult problem than that of finding clusters within the subgraph $\Delta G(t_1, t_2)$. Methods for finding such subgraphs are proposed in [20].

12.5.3.3 GraphScope

GraphScope [519] is a method to detect significant changes in bipartite (directed) graphs. Such graphs are useful for modeling a wide variety of directional interactions, such as users with Web sites, clients with hosts, and so on. Furthermore, since these interactions are dynamic, the patterns in the interactions will change over time.

The work in [519] uses a combination of dynamic community detection and information-theoretic methods to identify the key change points. In particular, the Minimum Description Length (MDL) principle is leveraged to identify change points. Intuitively, a change point significantly increases the encoding cost (MDL cost) to represent the stream. Therefore, by computing the change in the MDL cost, one can detect key change points.

Given a set of temporal snapshots $G_1 \ldots G_t$, the idea is to group them into *contiguous segments*, which are separated by change points (denoted by '$*$' below):

$$\underbrace{\{G_1, G_2, G_3\}}_{\text{Segment 1}}, * \underbrace{\{G_4, G_5\}}_{\text{Segment 2}} * \underbrace{\{G_6, G_7, G_8, G_9\}}_{\text{Segment 3}} * \ldots$$

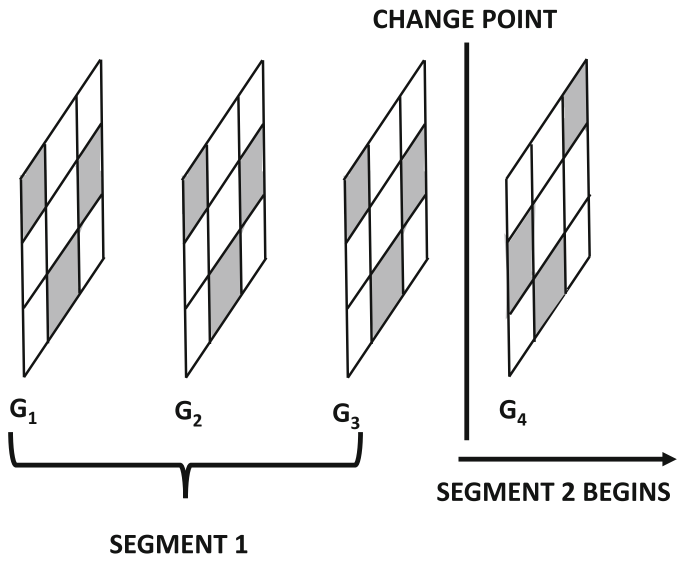

Figure 12.3: Illustrating change points in *GraphScope*

Each segment is compactly stored (i.e., encoded) in terms of the best fitting community structure of that segment, as well as the offsets (i.e., errors) of the individual graphs in that segment with respect to this best fit. The idea here is that graphs $G_1 \ldots G_3$ are very similar in terms of community structure (and can therefore be summarized in a small number of bits), but adding G_4 to this segment will greatly increase the encoding cost. This situation is illustrated in Figure 12.3. The main problem in *GraphScope* is to identify the best segmentation of the time-series of graphs, and also determine the best community structure within each segment. Note that one can find a common community structure in each segment only if the communities have not changed very significantly in that segment.

The approach groups similar sources together into source groups, and similar destinations together into destination groups. One can view this as a block-wise co-clustering, shared by all the snapshots of a segment, which is similar to the block-wise clustering in Figure 12.1. The main difference from Figure 12.1 is that there are no node overlaps among clusters. If the underlying communities do not change much over time, then the snapshot of the evolving graphs will have similar descriptions and can also be grouped together into a time segment, to achieve better compression. A binary Huffman encoding of the community structure within a segment is used in order to represent it. In principle, one could use any other type of matrix compression, such as a shared matrix factorization of all graphs in the segment, as long as the size and error of the compressed description can be properly encoded into a cost. Whenever a new graph snapshot cannot fit well into the old segment in terms of this description, *GraphScope* introduces a change point and starts a new segment. These change points are identified by comparing the cost of including the snapshot in the current segment versus starting a new segment. The lower cost option is always selected. It has been shown in [519], that such change points correspond to drastic discontinuities in the network, which can be regarded as temporal outliers. Readers are referred to [519] for details.

12.5.4 Outliers Based on Shortest Path Distance Changes

Most real-world graphs such as the Web, social networks and information networks experience significant changes in terms of the pairwise distances between nodes in the network. For example, it has been shown in [67] that most real graphs such as the Web and social networks have shrinking diameters over time. This is because edges are continuously added to such networks, which makes them more dense.

In this context, *sudden and abrupt* changes in pairwise distances between nodes are indicative of unusual events in a network. For example, in a bibliographic network such as *DBLP* [625], most pairs of nodes within a specific topical area can typically be connected by small paths of lengths 2 or 3. On the other hand, the sudden addition of an edge that connects a pair of nodes at a distance of 5 is an unusual event. Most likely, this event reflects the sudden collaboration between a pair of authors in different topical areas. Such changes can, therefore, provide useful insights.

A straightforward solution to this problem is to solve the all-pairs shortest-path problem [40] at two snapshots t_1 and t_2. The pairs of nodes for which the distances have changed very significantly are reported as the anomalies. However, such an algorithm requires the computation and storage of all-pairs shortest paths, which can be impractical in large-scale settings, such as social, communication, information and Web networks. For example, in a network containing 10^8 nodes, the number of possible pairs is 10^{16}. The computation and storage of such a large number of node pairs becomes cumbersome and impractical.

Therefore, it is important to design methods that can *efficiently* identify the top-k distance changes in a heuristic way. A key observation [234] is that edges on shortest paths between many pairs of nodes in either snapshot are important edges. The addition or deletion of such edges can significantly change the shortest path distances. Therefore, a randomized algorithm is proposed in [234] to identify such edges. This is then leveraged in order to identify significant node pairs between which the greatest change has occurred. Although the approach is heuristic, high precision and recall are achieved.

12.5.5 Matrix Factorization and Latent Embedding Methods

Many methods in the literature discover latent embeddings over sequences of graph snapshots, while incorporating temporal smoothness constraints [143, 598]. Violations of temporal smoothness constraints represent change points. Unlike the hotspot approach with PCA (cf. section 12.5.1), this approach does require the snapshot-based adjacency matrices $A_1 \dots A_T$ and is therefore not designed for a fully streaming scenario. The total number of snapshots is denoted by T. In the following, we discuss the general principle of this method without focusing on any particular technique. Therefore, the description below is more general and quite different from that discussed in [143, 598].

The basic idea is to assume that each adjacency matrix A_t at time stamp t can be factorized into two matrices U_t and V_t:

$$A_t \approx U_t V_t^T \quad \forall t \in \{1 \dots T\} \tag{12.13}$$

Furthermore, we incorporate the temporal smoothness constraints to ensure that the latent structure does not change suddenly over successive snapshots:

$$\begin{aligned} U_t &\approx U_{t+1} \quad \forall t \in \{1 \dots T-1\} \\ V_t &\approx V_{t+1} \quad \forall t \in \{1 \dots T-1\} \end{aligned}$$

All these constraints can be incorporated into a single objective function with balancing parameters α and β as follows:

$$\text{Minimize } J = \underbrace{\sum_{t=1}^{T} ||A_t - U_t V_t^T||^2}_{\text{Embedding Objective}} + \underbrace{\alpha \sum_{t=1}^{T-1} ||U_t - U_{t+1}||^2 + \beta \sum_{t=1}^{T-1} ||V_t - V_{t+1}||^2}_{\text{Temporal Smoothness Regularization}}$$

Time-stamps at which the last two terms are large represent change points. Furthermore, by examining the contributions of individual nodes to the last two terms, one can also identify the locality of the changes. Furthermore, the residuals in $A_t - U_t V_t^T$ provide temporally inconsistent edges at time t. Therefore, this framework in very general in providing insights about different types of change points. This approach also has the merit of being applicable to directed graphs.

One can change the aforementioned objective function in various ways. For example, instead of penalizing U_t for violating temporal smoothness, one might fix each U_t to the same value. However, V_t is allowed to vary. This results in a shared matrix factorization. Adding regularizers corresponding to the Frobenius norms of U_t and V_t reduces overfitting. In other words, the term $\lambda \cdot \sum_{t=1}^{T}(||U_t||^2 + ||V_t||^2)$ can be added to the aforementioned objective function to reduce overfitting. Different types of regularizers would result in different properties of the embedding. One can also incorporate nonnegativity constraints to make the solution more interpretable. Nonnegative matrix factorization methods yield latent representations that are similar to temporal variants of Probabilistic Latent Semantic Analysis (PLSA). The main difference is that the objective function of non-negative matrix factorization uses the Frobenius norm, whereas PLSA uses maximum likelihood maximization of a generative model. As discussed in Chapter 8, such factorizations have the merit of probabilistic interpretability. Adding orthogonality constraints on the columns of U_t and V_t leads to embeddings similar to PCA and SVD. Such a factorization often has better geometric interpretability and out-of-sample generalizability because of the ability to easily project a point into the different orthogonal directions. These different options provide useful insights in different scenarios. In general, one should choose a specific type of factorization after carefully examining the needs of the application at hand.

It is also possible to combine the approach with the content-centric shared matrix factorization method of section 12.4.1 in order to address cases in which attributes are associated with the nodes at various snapshots. In such cases, one can model co-evolving network structure and attributes. Many social and Web networks have rich attributes associated with the nodes in the network. In such cases, the attributes can greatly help in modeling the network evolution process. In general, latent embedding methods provide the widest choices in enabling different types of application-centric scenarios by changing the corresponding objective function.

12.6 Conclusions and Summary

The problem of network outlier detection is particularly important because of the ubiquity of different kinds of graphs and networks in a wide variety of problem domains. Graphs can either occur as multiple entities of small size, or as a single large graph. Furthermore, since the nature of graphs is complex, the outliers can be defined in a wide variety of ways, depending upon how regularity is defined. Outliers can be defined in terms of nodes, edges, subgraphs, evolving edges, evolving subgraphs, or evolving distances. Therefore, it is critical to use application-specific properties to define network outliers in a meaningful way.

Most of the network outlier detection methods examine community changes in the underlying network in one form or the other. This can be done directly using community detection methods, or indirectly through edge sample and spectral methods. Many of these methods have also been generalized to the temporal context. Furthermore, it is relatively easy to generalize such methods to the content-centric scenario with the use of matrix factorization.

12.7 Bibliographic Survey

Several surveys have recently been written in the context of graph anomaly detection. A general survey on graph-based anomaly detection and description is provided in [43]. Surveys have also been written on anomaly detection in the temporal context. A recent survey on evolutionary network analysis is provided in [14]. Anomaly detection in dynamic networks is discussed in [457]. Detailed surveys on outlier detection in temporal data may be found in [231, 232].

Outlier detection can be defined in the context of many small graphs, or in the context of a single large graph. The former case occurs frequently in scenarios such as chemical and XML data, in which unusual objects need to be determined [15, 558], Two common methods can be used. Distance-based methods can be easily generalized to this case [558], by defining appropriate similarity functions between the graphs. Alternatively, clustering or frequent pattern mining methods [15] can be used to identify significant anomalies.

Outliers in graphs can be defined in the form of node outliers, linkage outliers, or subgraph outliers. The wide variety of ways in which outliers can be defined in networks is a direct result of the complexity of the data. Even in the context of particular kinds of outliers such as node outliers [41], it has been shown that a wide variety of definitions are possible, depending upon how the features are defined within the locality of a node. A network embedding approach to finding anomalies is discussed in [277]. Linkage outliers are defined as edges that lie across dense clusters in networks [17]. Sketch-based methods for anomaly detection in streaming heterogeneous graphs and edge streams are discussed in [386, 458]. The basic idea is to create a vector representation from the local substructures of a graph for fast similarity computation. The local substructures are themselves represented as short strings.

A method for finding linkage outliers with the use of the Minimum Description Length (MDL) principle is proposed in [120]. The MDL principle is also useful for finding subgraph outliers [416]. Other work along a similar direction was presented in [179]. The use of eigenvector analysis to compare subgraphs with background behavior of the full graph and correspondingly declare them as anomalous was proposed in [397]. The use of such an approach for threat detection in social networks for applications such as counter-terrorism is discussed in [398].

In many practical scenarios, content is available at the nodes. Some examples of such scenarios include social networks, information networks, and the Web. Such content can be used to significantly enhance and sharpen the outlier analysis process. The work in [212] uses the similarity between the features at a node and its neighbors to determine linkage outliers. The work in [452] determines outlier links simultaneously with the underlying communities, with the use of a heterogeneous Markov random field (HMRF) approach. Finally, the work in [214] determines community outliers, which are sets of linked nodes that are mutually inconsistent with the underlying content. The reverse problem of determining attribute outliers by examining the hierarchical structure of data such as XML has been addressed in [320]. This is because the hierarchical structure of XML data provides semantically meaningful neighborhoods in which outliers may be found. The *FocusCO* algorithm [433] discovers clusters and outliers simultaneously in attributed graphs with the help of user supervision. Scalable methods for anomaly detection in attributed networks are discussed in [434].

Significant research has also been focused on leveraging user supervision for detecting outliers in networks. User supervision is particularly important in the network domain because of the inherent complexity of network data. The aforementioned *FocusCO* is an algorithm that leverages user supervision [433]. Another approach for user supervision is to allow the use of *query templates*, in which the users can determine outlier subgraphs with specific properties [237, 238].

An important problem is that of temporal outlier detection in evolutionary graphs [14]. Outliers can be defined in an almost unlimited number of ways in temporal graphs, because of the different combinations of time and structure, which can be used in order to define regularity. Some of the earliest work focuses on measuring similarities between successive snapshots of graphs with the use of different similarity functions [438]. Another method that uses graph matching between successive snapshots for anomaly detection is discussed in [492]. This creates a time-series which can be analyzed with standard auto-regressive moving average (ARMA) methods for finding the outliers. The work in [280] uses spectral methods to determine anomalies in time-series of graphs. The principal component is chosen as the activity vector for that graph. This graph is then represented as a time series of activity vectors, which creates a data set of activity vector values. The principal left singular vector of this data set provides the significant direction of correlation. The activity vector for a new graph is computed, and the corresponding angle with the principal left singular vector provides the outlier score. A method to detect node anomalies in graph streams with eigenvector analysis is discussed in [597].

The work in [522] uses compact matrix decomposition to approximate the adjacency matrix of large sparse graphs. The primary idea underlying the work is that it is harder to approximate anomalous graphs than normal graphs. Therefore, the approximation error for each graph in a sequence of graphs is constructed. Anomaly detection is performed on this time-series of values. The use of tensor analysis to discover comet communities is proposed in [61]. Other dimensionality reduction methods that are used for anomaly detection in graphs include the use of non-negative matrix factorization [551].

The use of community evolution methods is very common, since communities capture the broad patterns in the network. Therefore, a change in the community structure is used to model significant evolution [20, 143, 233, 235, 236, 383, 519, 520, 521]. The work in [449] uses the history of the node's neighborhood to detect anomalies. Other common methods used are shortest-path distance change metrics [234], and latent embedding methods [143, 598]. Some methods [519, 521] are specifically applicable to bipartite graphs. The determination of significant evolution in graphs can be useful in the context of a wide variety of applications such as monitoring blog communities [415], or mining traffic flow data sets [400]. In the latter case, values are associated with edges, corresponding to traffic flows. Anomalous regions are found in the network, by using the values on these edges. Incremental anomaly detection with the use of *commute time* is discussed in [314]. The basic idea is that incoming nodes with large commute times to existing nodes in the network are reported as anomalies. The commute time is a measure of the distance between two nodes, and it is equal to the expected number of hops required by a random walk to travel from one node to another and back. Therefore, the approach is a proximity-based outlier detection method, and it is related to incremental community maintenance methods. The main innovation in this work is the ability to compute the commute times in incremental fashion in an efficient way.

Finally, many forms of pattern changes in a network may be characterized in the form of evolution rules. In the framework presented in [81], nodes, and edges are associated with labels (i.e., specific properties of the network). Furthermore, the time-stamps of the first appearance of edges are maintained. Patterns are defined as subgraphs that have similar structure and labels on nodes at different time stamps, and the same relative offsets of the time-stamps. This defines significant temporal patterns or *graph evolution rules* in the underlying data. Evolution rules do not necessarily represent outliers, since they correspond to frequent temporal patterns in the data. On the other hand, the formation of a new evolution rule at a given time may be considered a temporal novelty and may be reported as an outlier.

In many evolutionary graphs, a significant amount of content may be available. Some examples of such data include social streams such as those created by *Twitter*. Therefore, anomaly detection in such scenarios may need to combine both linkage and content. A number of recent methods have also shown how to determine evolutionary outliers in social media streams [30, 435, 530]. Some methods focus on linkage [530], whereas others focus on content [435], and others on a combination of the two [30]. The detection of anomalous meetings between a group of people in a social network is addressed in [495] with the use of an expectation-maximization approach.

12.8 Exercises

1. Download the *DBLP* bibliographic network [625]. Construct the co-authorship network, where the weights on the edges correspond to the number of publications. Extract the features n_i, e_i, w_i and λ_i, as discussed in the node outlier section of this chapter. Create pairwise plots between:

- n_i and e_i
- w_i and e_i
- λ_i and w_i

Determine the points which deviate significantly from the least squares fit for each pair. Which nodes are determined as the outliers?

2. Use an edge sampling approach in order to create k connected components in the network, where $k = 100000$. Repeat the process 100 times. Which edges have end points that repeatedly lie in different partitions?

3. Construct a *DBLP* co-authorship network for each year since 1990. Construct a normalized dot-product between the edge sets in successive years. Which are the significant change points along the different years?

4. Repeat Exercise 3 by performing PCA on the un-weighted augmented adjacency matrix graph snapshot obtained from each year of the *DBLP* data set. PCA is used to obtain each year's multidimensional representation on which the dot product is performed.

5. Associate a weight with each edge of the *DBLP* network, corresponding to the number of publications between that author pair. Repeat Exercise 4 using these weighted edges.