Chapter 19

Social Network Analysis

"*I hope we will use the Net to cross barriers and connect cultures.*"—Tim Berners-Lee

19.1 Introduction

The tendency of humans to connect with one another is a deep-rooted social need that precedes the advent of the Web and Internet technologies. In the past, social interactions were achieved through face-to-face contact, postal mail, and telecommunication technologies. The last of these is also relatively recent when compared with the history of mankind. However, the popularization of the Web and Internet technologies has opened up entirely new avenues for enabling the seamless interaction of geographically distributed participants. This extraordinary potential of the Web was observed during its infancy by its visionary founders. However, it required a decade before the true social potential of the Web could be realized. Even today, Web-based social applications continue to evolve and create an ever-increasing amount of data. This data is a treasure trove of information about user preferences, their connections, and their influences on others. Therefore, it is natural to leverage this data for analytical insights.

Although social networks are popularly understood in the context of large online networks such as Twitter, LinkedIn, and Facebook, such networks represent only a small minority of the interaction mechanisms enabled by the Web. In fact, the traditional study of social network analysis in the field of sociology precedes the popularization of technologically enabled mechanisms. Much of the discussion in this chapter applies to social networks that extend beyond the popular notions of online social networks. Some examples are as follows:

- Social networks have been studied extensively in the field of sociology for more than a century but not from an online perspective. Data collection was rather difficult in these scenarios because of the lack of adequate technological mechanisms. Therefore, these studies were often conducted with painstaking and laborious methods for manual data collection. An example of such an effort is Stanley Milgram's famous six degrees of separation experiment in the sixties, which used postal mail between participants

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_19

to test whether two arbitrary humans on the planet could be connected by a chain of six relationships. Because of the difficulty in verifying local forwards of mail, such experiments were often hard to conduct in a trustworthy way. Nevertheless, in spite of the obvious flaws in the experimental setting, these results have recently been shown to be applicable to online social networks, where the relationships between individuals are more easily quantifiable.

- A number of technological enablers, such as telecommunications, email, and electronic chat messengers, can be considered indirect forms of social networks. Such enablers result in communications between different individuals, and therefore they have a natural social aspect.

- Sites that are used for sharing online media content, such as Flickr, YouTube, or Delicious, can also be considered indirect forms of social networks, because they allow an extensive level of user interaction. In addition, social media outlets provide a number of unique ways for users to interact with one another. Examples include posting blogs or tagging each other's images. In these cases, the interaction is centered around a specific service such as content-sharing; yet many fundamental principles of social networking apply. Such social networks are extremely rich from the perspective of mining applications. They contain a tremendous amount of content such as text, images, audio, or video.

- A number of social networks can be constructed from specific kinds of interactions in professional communities. Scientific communities are organized into bibliographic and citation networks. These networks are also content rich because they are organized around publications.

It is evident that these different kinds of networks illustrate different facets of social network analysis. Many of the fundamental problems discussed in this chapter apply to these different scenarios but in different settings. Most of the traditional problems in data mining, such as clustering and classification, can also be extended to social network analysis. Furthermore, a number of more complex problem definitions are possible, such as link prediction and social influence analysis, because of the greater complexity of networks as compared to other kinds of data.

This chapter is organized as follows. Section 19.2 discusses a number of fundamental properties of social network analysis. The problem of community detection is explained in Sect. 19.3. The collective classification problem is discussed in Sect. 19.4. Section 19.5 discusses the link prediction problem. The social influence analysis problem is addressed in Sect. 19.6. The chapter summary is presented in Sect. 19.7.

19.2 Social Networks: Preliminaries and Properties

It is assumed that the social network can be structured as a graph $G = (N, A)$, where N is the set of nodes and A is the set of edges. Each individual in the social network is represented by a node in N, and is also referred to as an *actor*. The edges represent the connections between the different actors. In a social network such as Facebook, these edges correspond to friendship links. Typically, these links are undirected, although it is also possible for some "follower-based" social networks, such as Twitter, to have directed links. By default, it will be assumed that the network $G = (N, A)$ is undirected, unless otherwise specified. In some cases, the nodes in N may have content associated with them. This content may

correspond to comments or other documents posted by social network users. It is assumed that the social network contains n nodes and m edges. In the following, some key properties of social networks will be discussed.

19.2.1 Homophily

Homophily is a fundamental property of social networks that is used in many applications, such as node classification. The basic idea in homophily is that nodes that are connected to one another are more likely to have similar properties. For example, a person's friendship links in Facebook may be drawn from previous acquaintances in school and work. Aside from common backgrounds, the friendship links may often imply common interests between the two parties. Thus, individuals who are linked may often share common beliefs, backgrounds, education, hobbies, or interests. This is best stated in terms of the old proverb:

Birds of a feather flock together

This property is leveraged in many network-centric applications.

19.2.2 Triadic Closure and Clustering Coefficient

Intuitively, triadic closure may be thought of as an inherent tendency of real-world networks to cluster. The principle of triadic closure is as follows:

If two individuals in a social network have a friend in common, then it is more likely that they are either connected or will eventually become connected in the future.

The principle of triadic closure implies an inherent correlation in the edge structure of the network. This is a natural consequence of the fact that two individuals connected to the same person are more likely to have similar backgrounds and also greater opportunities to interact with one another. The concept of triadic closure is related to homophily. Just as the similarity in backgrounds of connected individuals makes their properties similar, it also makes it more likely for them to be connected to the same set of actors. While homphily is typically exhibited in terms of content properties of node attributes, triadic closure can be viewed as the structural version of homophily. The concept of triadic closure is directly related to the *clustering coefficient* of the network.

The *clustering coefficient* can be viewed as a measure of the inherent tendency of a network to cluster. This is similar to the Hopkins statistic for multidimensional data (cf. Sect. 6.2.1.4 of Chap. 6). Let $S_i \subseteq N$ be the set of nodes connected to node $i \in N$ in the undirected network $G = (N, A)$. Let the cardinality of S_i be n_i. There are $\binom{n_i}{2}$ possible edges between nodes in S_i. The local clustering coefficient $\eta(i)$ of node i is the fraction of these pairs that have an edge between them.

$$\eta(i) = \frac{|\{(j,k) \in A : j \in S_i, k \in S_i\}|}{\binom{n_i}{2}} \tag{19.1}$$

The Watts–Strogatz *network average clustering coefficient* is the average value of $\eta(i)$ over all nodes in the network. It is not difficult to see that the triadic closure property increases the clustering coefficient of real-world networks.

19.2.3 Dynamics of Network Formation

Many real properties of networks are affected by how they are formed. Networks such as the World Wide Web and social networks are continuously growing over time with new nodes and edges being added constantly. Interestingly, networks from multiple domains share a number of common characteristics in the dynamic processes by which they grow. The manner in which new edges and nodes are added to the network has a direct impact on the eventual structure of the network and choice of effective mining techniques. Therefore, the following will discuss some common properties of real-world networks:

1. *Preferential attachment:* In a growing network, the likelihood of a node receiving new edges increases with its degree. This is a natural consequence of the fact that highly connected individuals will typically find it easier to make new connections. If $\pi(i)$ is the probability that a newly added node attaches itself to an existing node i in the network, then a model for the probability $\pi(i)$ in terms of the degree of node i is as follows:

$$\pi(i) \propto \text{Degree}(i)^{\alpha} \tag{19.2}$$

   The value of the parameter α is dependent on the domain from which the network is drawn, such as a biological network or social network. In many Web-centric domains, a *scale-free assumption* is used. This assumption states that $\alpha \approx 1$, and therefore the proportionality is linear. Such networks are referred to as *scale-free* networks. This model is also referred to as the *Barabasi–Albert model*. Many networks, such as the World Wide Web, social networks, and biological networks, are conjectured to be scale free, although the assumption is obviously intended to be an approximation. In fact, many properties of real networks are not completely consistent with the scale-free assumption.

2. *Small world property:* Most real networks are assumed to be "small world." This means that the average path length between any pair of nodes is quite small. In fact, Milgram's experiment in the sixties conjectured that the distance between any pair of nodes is about six. Typically, for a network containing $n(t)$ nodes at time t, many models postulate that the average path lengths grow as $\log(n(t))$. This is a small number, even for very large networks. Recent experiments have confirmed that the average path lengths of large-scale networks such as Internet chat networks are quite small. As discussed below, the dynamically varying diameters have been experimentally shown to be even more constricted than the (modeled) $\log(n(t))$ growth rate would suggest.

3. *Densification:* Almost all real-world networks such as the Web and social networks add more nodes and edges over time than are deleted. The impact of adding new edges generally dominates the impact of adding new nodes. This implies that the graphs gradually densify over time, with the number of edges growing superlinearly with the number of nodes. If $n(t)$ is the number of nodes in the network at time t, and $e(t)$ is the number of edges, then the network exhibits the following *densification power law*:

$$e(t) \propto n(t)^{\beta} \tag{19.3}$$

   The exponent β is a value between 1 and 2. The value of $\beta = 1$ corresponds to a network where the average degree of the nodes is not affected by the growth of the network. A value of $\beta = 2$ corresponds to a network in which the total number of

edges $e(t)$ remains a constant fraction of the complete graph of $n(t)$ nodes as $n(t)$ increases.

4. *Shrinking diameters:* In most real-world networks, as the network densifies, the average distances between the nodes shrink over time. This experimental observation is in contrast to conventional models that suggest that the diameters should increase as $\log(n(t))$. This unexpected behavior is a consequence of the fact that the addition of new edges dominates the addition of new nodes. Note that if the impact of adding new nodes were to dominate, then the average distances between nodes would increase over time.

5. *Giant connected component:* As the network densifies over time, a giant connected component emerges. The emergence of a giant connected component is consistent with the principle of preferential attachment, in which newly incoming edges are more likely to attach themselves to the densely connected and high-degree nodes in the network. This property also has a confounding impact on network clustering algorithms, because it typically leads to unbalanced clusters, unless the algorithms are carefully designed.

Preferential attachment also has a significant impact on the typical structure of online networks. It results in a small number of very high-degree nodes that are also referred to as *hubs*. The hub nodes are usually connected to many different regions of the network and, therefore, have a confounding impact on many network clustering algorithms. The notion of hubs, as discussed here, is subtly different from the notion of hubs, as discussed in the *HITS* algorithm, because it is not specific to a query or topic. Nevertheless, the intuitive notion of nodes being central points of connectivity in a network, is retained in both cases.

19.2.4 Power-Law Degree Distributions

A consequence of preferential attachment is that a small minority of high-degree nodes continue to attract most of the newly added nodes. It can be shown that the number of nodes $P(k)$ with degree k, is regulated by the following *power-law degree distribution*:

$$P(k) \propto k^{-\gamma} \tag{19.4}$$

The value of the parameter γ ranges between 2 and 3. It is noteworthy that larger values of γ lead to more small degree nodes. For example, when the value of γ is 3, the vast majority of the nodes in the network will have a degree of 1. On the other hand, when the value of γ is small, the degree distribution is less skewed.

19.2.5 Measures of Centrality and Prestige

Nodes that are central to the network have a significant impact on the properties of the network, such as its density, pairwise shortest path distances, connectivity, and clustering behavior. Many of these nodes are hub nodes, with high degrees that are a natural result of the dynamical processes of large network generation. Such actors are often more prominent because they have ties to many actors and are in a position of better influence. Their impact on network mining algorithms is also very significant. A related notion of centrality is *prestige*, which is relevant for directed networks. For example, on Twitter, an actor with a larger number of followers has greater prestige. On the other hand, following a large number of individuals does not bring any prestige but is indicative of the *gregariousness* of an actor.

The notion of *PageRank*, discussed in the previous chapter, is often used as a measure of prestige.

Measures of centrality are naturally defined for undirected networks, whereas measures of prestige are designed for directed networks. However, it is possible to generalize centrality measures to directed networks. In the following, centrality measures will be defined for undirected networks, whereas prestige measures will be defined for directed networks.

19.2.5.1 Degree Centrality and Prestige

The degree centrality $C_D(i)$ of a node i of an undirected network is equal to the degree of the node, divided by the maximum *possible* degree of the nodes. The maximum possible degree of a node in the network is one less than the number of nodes in the network. Therefore, if Degree(i) is the degree of node i, then the degree centrality $C_D(i)$ of node i is defined as follows:

$$C_D(i) = \frac{\text{Degree}(i)}{n-1} \tag{19.5}$$

Because nodes with higher degree are often hub nodes, they tend to be more central to the network and bring distant parts of the network closer together. The major problem with degree centrality is that it is rather myopic in that it does not consider nodes beyond the immediate neighborhood of a given node i. Therefore, the overall structure of the network is ignored to some extent. For example, in Fig. 19.1a, node 1 has the highest degree centrality, but it cannot be viewed as central to the network itself. In fact, node 1 is closer to the periphery of the network.

Degree *prestige* is defined for directed networks only, and uses the indegree of the node, rather than its degree. The idea is that only a high indegree contributes to the prestige because the indegree of a node can be viewed as a vote for the popularity of the node, similar to *PageRank*. Therefore, the degree prestige $P_D(i)$ of node i is defined as follows:

$$P_D(i) = \frac{\text{Indegree}(i)}{n-1} \tag{19.6}$$

For example, node 1 has the highest degree prestige in Fig. 19.1b. It is possible to generalize this notion recursively by taking into account the prestige of nodes pointing to a node, rather than simply the number of nodes. This corresponds to the rank prestige, which will be discussed later in this section.

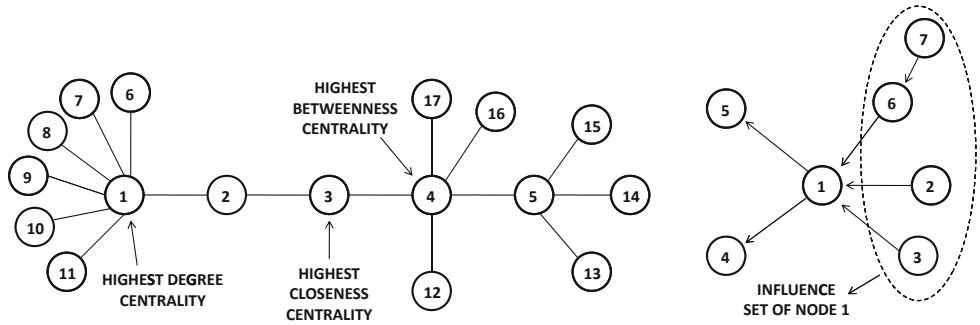
The notion of centrality can also be extended to the node *outdegree*. This is defined as the *gregariousness* of a node. Therefore, the gregariousness $G_D(i)$ of a node i is defined as follows:

$$G_D(i) = \frac{\text{Outdegree}(i)}{n-1} \tag{19.7}$$

The gregariousness of a node defines a different qualitative notion than prestige because it quantifies the propensity of an individual to seek out new connections (such as following many other actors in Twitter), rather than his or her popularity with respect to other actors.

19.2.5.2 Closeness Centrality and Proximity Prestige

The example of Fig. 19.1a shows that the degree centrality criterion is susceptible to picking nodes on the periphery of the network with no regard to their *indirect* relationships to other nodes. In this context, closeness centrality is more effective.



(a) centrality illustration (b) proximity prestige

Figure 19.1: Illustration of centrality and prestige

The notion of closeness centrality is meaningfully defined with respect to *undirected* and *connected* networks. The average shortest path distance, starting from node i, is denoted by $\text{AvDist}(i)$ and is defined in terms of the pairwise shortest path distances $\text{Dist}(i, j)$, between nodes i and j as follows:

$$\text{AvDist}(i) = \frac{\sum_{j=1}^{n} \text{Dist}(i, j)}{n - 1} \tag{19.8}$$

The closeness centrality is simply the inverse of the average distance of other nodes to node i.

$$C_C(i) = 1/\text{AvDist}(i) \tag{19.9}$$

Because the value of $\text{AvDist}(i)$ is at least 1, this measure ranges between 0 and 1. In the case of Fig. 19.1a, node 3 has the highest closeness centrality because it has the lowest average distance to other nodes.

A measure known as *proximity prestige* can be used to measure prestige in directed networks. To compute the proximity prestige of node i, the shortest path distance to node i from all other nodes is computed. Unlike undirected networks, a confounding factor in the computation is that *directed* paths may not exist from other nodes to node i. For example, no path exists to node 7 in Fig. 19.1b. Therefore, the first step is to determine the set of nodes $\text{Influence}(i)$ that can *reach* node i with a directed path. For example, in the case of the Twitter network, $\text{Influence}(i)$ corresponds to all recursively defined followers of node i. An example of an influence set of node 1 is illustrated in Fig. 19.1b. The value of $\text{AvDist}(i)$ can now be computed only with respect to the influence set $\text{Influence}(i)$.

$$\text{AvDist}(i) = \frac{\sum_{j \in \text{Influence}(i)} \text{Dist}(j, i)}{|\text{Influence}(i)|} \tag{19.10}$$

Note that distances are computed from node j to i, and not vice versa, because we are computing a prestige measure, rather than a gregariousness measure.

Both the size of the influence set and average distance to the influence set play a role in defining the proximity prestige. While it is tempting to use the inverse of the average distance, as in the previous case, this would not be fair. Nodes that have less influence should be penalized. For example, in Fig. 19.1b, node 6 has the lowest possible distance value of 1 from node 7, which is also the only node it influences. While its low average distance to its influence set suggests high prestige, its small influence set suggests that it

cannot be considered a node with high prestige. To account for this, a multiplicative penalty factor is included in the measure that corresponds to the fractional size of the influence set of node i.

$$\text{InfluenceFraction}(i) = \frac{|\text{Influence}(i)|}{n-1} \tag{19.11}$$

Then, the proximity prestige $P_P(i)$ is defined as follows:

$$P_P(i) = \frac{\text{InfluenceFraction}(i)}{\text{AvDist}(i)} \tag{19.12}$$

This value also lies between 0 and 1. Higher values indicate greater prestige. The highest possible proximity prestige value of 1 is realized at the central node of a perfectly star-structured network, with a single central actor and all other actors as its (in-linking) spokes.

In the case of Fig. 19.1b, node 1 has an influence fraction of 4/6, and an average distance of 5/4 from the four nodes that reach it. Therefore, its proximity prestige is $4 * 4/(5 * 6) = 16/30$. On the other hand, node 6 has a better average distance of 1 to the only node that reaches it. However, because its influence fraction is only 1/6, its proximity prestige is 1/6 as well. This suggests that node 1 has better proximity prestige than node 6. This matches our earlier stated intuition that node 6 is not a very influential node.

19.2.5.3 Betweenness Centrality

While closeness centrality is based on notions of distances, it does not account for the *criticality* of the node in terms of the number of shortest paths that pass through it. Such notions of criticality are crucial in determining actors that have the greatest control of the flow of information between other actors in a social network. For example, while node 3 has the highest closeness centrality, it is not as critical to shortest paths between different pairs of nodes as node 4 in Fig. 19.1a. Node 4 can be shown to be more critical because it also participates in shortest paths between the pairs of nodes directly incident on it, whereas node 3 does not participate in these pairs. The other pairs are approximately the same in the two cases. Therefore, node 4 controls the flow of information between nodes 12 and 17 that node 3 does not control.

Let q_{jk} denote the number of shortest paths between nodes j and k. For graphs that are not trees, there will often be more than one shortest path between pairs of nodes. Let $q_{jk}(i)$ be the number of these pairs that pass through node i. Then, the fraction of pairs $f_{jk}(i)$ that pass through node i is given by $f_{jk}(i) = q_{jk}(i)/q_{jk}$. Intuitively, $f_{jk}(i)$ is a fraction that indicates the level of control that node i has over nodes j and k in terms of regulating the flow of information between them. Then, the betweenness centrality $C_B(i)$ is the average value of this fraction over all $\binom{n}{2}$ pairs of nodes.

$$C_B(i) = \frac{\sum_{j<k} f_{jk}(i)}{\binom{n}{2}} \tag{19.13}$$

The betweenness centrality also lies between 0 and 1, with higher values indicating better betweenness. Unlike closeness centrality, betweenness centrality can be defined for disconnected networks as well.

While the aforementioned notion of betweenness centrality is designed for nodes, it can be generalized to edges by using the number of shortest paths passing through an edge (rather than a node). For example, the edges connected to the hub nodes in Fig. 19.2 have high betweenness. Edges that have high betweenness tend to connect nodes from different

clusters in the graph. Therefore, these betweenness concepts are used in many community detection algorithms, such as the Girvan–Newman algorithm. In fact, the computation of node- and edge-betweenness values is described in Sect. 19.4 on the Girvan–Newman algorithm.

19.2.5.4 Rank Centrality and Prestige

The concepts of rank centrality and prestige are defined by random surfer models. The *PageRank* score can be considered a rank centrality score in undirected networks and a rank prestige score in directed networks. Note that the *PageRank* scores are components of the largest left eigenvector of the random walk transition matrix of the social network. If the adjacency matrix is directly used instead of the transition matrix to compute the largest eigenvector, the resulting scores are referred to as *eigenvector centrality* scores. Eigenvector centrality scores are generally less desirable than *PageRank* scores because of the disproportionately large influence of high-degree nodes on the centrality scores of their neighbors.

Because the computation of these scores was already discussed in detail in Chap. 18, it will not be revisited here. The idea here is that a citation of a node by another node (such as a follower in Twitter) is indicative of prestige. Although this is also captured by degree prestige, the latter does not capture the prestige of the nodes incident on it. The *PageRank* computation can be considered a refined version of degree prestige, where the *quality* of the nodes incident on a particular node i are used in the computation of its prestige.

19.3 Community Detection

The term "community detection" is an approximate synonym for "clustering" in the context of social network analysis. The clustering of networks and graphs is also sometimes referred to as "graph partitioning" in the traditional work on network analysis. Therefore, the literature in this area is rich and includes work from many different fields. Much of the work on graph partitioning precedes the formal study of social network analysis. Nevertheless, it continues to be relevant to the domain of social networks. Community detection is one of the most fundamental problems in social network analysis. The summarization of closely related social groups is, after all, one of the most succinct and easily understandable ways of characterizing social structures.

In the social network domain, network clustering algorithms often have difficulty in cleanly separating out different clusters because of some natural properties of typical social networks.

- Multidimensional clustering methods, such as the distance-based k-means algorithm, cannot be easily generalized to networks. In small-world networks, the distances between different pairs of nodes is a small number that cannot provide a sufficiently fine-grained indicator of similarity. Rather, it is more important to use triadic closure properties of real networks, explicitly or implicitly, in the clustering process.
- While social networks usually have distinct community structures, the high-degree hub nodes connect different communities, thereby bringing them together. Examples of such hub nodes connecting up different communities are illustrated in Fig. 19.2. In this case, the nodes A, B, and C are hubs that connect up different communities. In real social networks, the structure may be even more complicated, with some of the high-degree nodes belonging to particular sets of overlapping communities.

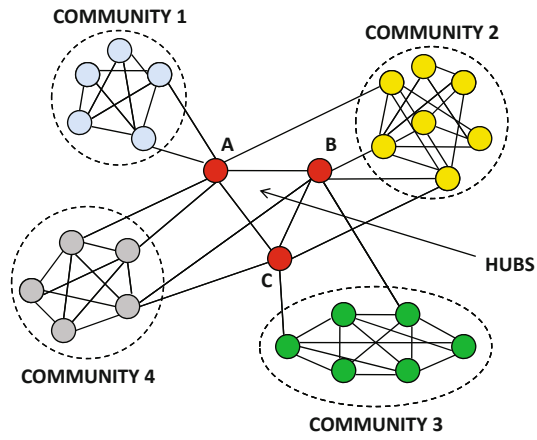


Figure 19.2: Impact of hubs on communities

- Different parts of the social network have different edge densities. In other words, the local clustering coefficients in distinct parts of the social network are typically quite different. As a result, when specific choices of parameters are used to quantify the clusters globally, it leads to unbalanced clusters because a single global parameter choice is not relevant in many network localities.
- Real social networks often have a giant component that is densely connected. This contributes further to the tendency of community detection algorithms to create imbalanced clusters, where a single cluster is the beneficiary of most of the nodes in the network.

Many network clustering algorithms have built-in mechanisms to address such issues. In the following, a discussion of some of the most well-known network clustering algorithms will be provided.

Assume that the undirected network is denoted by $G = (N, A)$. The weight of the edge (i, j) between nodes i and j, is denoted by $w_{ij} = w_{ji}$. In some cases, the inverse concept of edge costs (or lengths) is specified instead of weights. In such cases, we assume that the edge cost is denoted by c_{ij}. These values can be converted to one another by using $w_{ij} = 1/c_{ij}$, or a suitably chosen kernel function.

The problem of network clustering, or community detection, is that of partitioning the network into k sets of nodes, such that the sum of the weights of the edges with end points in different partitions is minimized. Many variations of this basic objective function are used in practice and are able to achieve different application-specific goals, such as *partition balancing* in which different clusters have similar numbers of nodes.

In the special case, where $w_{ij} = 1$, and there are no balancing constraints on partitions, the 2-way cut problem is polynomially solvable. The reader is advised to refer to the bibliographic notes for pointers to Karger's randomized minimum cut algorithm. This algorithm can determine the minimum cut in $O(n^2 \log^r(n))$ time for a network containing n nodes, where r is a constant regulating the desired level of probabilistic accuracy. However, the resulting cut is usually not balanced. Incorporating arbitrary edge weights or balancing constraints makes the problem NP-hard. Many network clustering algorithms focus on balanced 2-way partitioning of graphs. A 2-way partitioning can be recursively used to generate a k-way partitioning.

19.3.1 Kernighan–Lin Algorithm

The Kernighan–Lin algorithm is a classical method for balanced 2-way graph partitioning. The basic idea is to start with an initial partitioning of the graph into two equal[1] subsets of nodes. The algorithm then iteratively improves this partitioning, until it converges to an optimal solution. This solution is not guaranteed to be the global optimum, but it is usually a good heuristic approximation. This iterative improvement is performed by determining sequences of exchanges of nodes between partitions that improve the clustering objective function as much as possible. To evaluate the improvement in the clustering objective function by performing an exchange between a pair of nodes, some carefully chosen measures need to be continuously tracked maintained at each node. These will be discussed below.

The *internal cost* I_i of node i is the sum of the weights of edges incident on i, whose other end is present in the same partition as node i. The *external cost* E_i of node i, is the sum of the weights of the edges incident on i, whose other end is in a different partition than node i. Moving a node from one partition to the other changes its external cost to its internal cost and vice versa. Therefore, the gain D_i by moving a node i from one partition to the other is given by the difference between the external and the internal cost.

$$D_i = E_i - I_i \tag{19.14}$$

Of course, we are not interested in simply moving a node from one partition to the other, but in exchanging a pair of nodes i and j between two partitions. Then, the gain J_{ij} of exchanging nodes i and j is given by the following:

$$J_{ij} = D_i + D_j - 2 \cdot w_{ij} \tag{19.15}$$

This is simply a sum of the gains from moving nodes i and j to different partitions, with a special adjustment for the impact on the edge (i, j) that continues to be a part of the external cost of both nodes because of the exchange. The value of J_{ij} therefore quantifies the gain that one can obtain by exchanging nodes i and j. Positive values of J_{ij} result in an improvement of the objective function.

The overall algorithm uses repeated sequences of at most $(n/2)$ heuristic exchanges between the two partitions, which are designed to optimize the total gain from the exchanges. Each such sequence of at most $(n/2)$ exchanges will be referred to as an *epoch*. Each epoch proceeds as follows. A pair of nodes is found, such that the exchange leads to the maximum improvement in the objective function value. This pair of nodes is marked, although the exchange is not actually performed. The values of D_i for different nodes are recomputed, however, as if the exchange were already performed. Then, the next pair of unmarked nodes is determined with these recomputed values of D_i, for which the exchange leads to the maximum improvement in the objective function value. It should be pointed out that the gains will not always decrease, as further potential exchanges are determined. Furthermore, some intermediate potential exchanges might even have negative gain, whereas later potential exchanges might have positive gain. The process of determining potential exchange pairs is repeated until all n nodes have been paired. Any sequence of $k \leq n/2$ contiguous potential pairs, starting with the first pair, and in the same order as they were determined, is considered a valid potential k-exchange between the two partitions. Among these different possibilities, the potential k-exchange maximizing the total gain is found. If the gain is positive, then the potential k-exchange is executed. This entire process of a

[1] Without loss of generality, it can be assumed that the graph contains an even number of nodes, by adding a single dummy node.

Algorithm *KernighanLin*(Graph: $G = (N, A)$, Weights:$[w_{ij}]$)
begin
  Create random initial partition of N into N_1 and N_2;
  repeat
    Recompute D_i values for each node $i \in N$;
    Unmark all nodes in N;
    for $i = 1$ to $n/2$ **do**
    begin
      Select $x_i \in N_1$ and $y_i \in N_2$ to be the unmarked node pair with
                the highest exchange-gain $g(i) = J_{x_i y_i}$;
      Mark x_i and y_i;
      Recompute D_j for each node j, under the assumption that
            x_i and y_i will be eventually exchanged;
    end
    Determine k that maximizes $G_k = \sum_{i=1}^{k} g(i)$;
    if $(G_k > 0)$ **then** exchange $\{x_1 \ldots x_k\}$ and
          $\{y_1 \ldots y_k\}$ between N_1 and N_2;
  until $(G_k \leq 0)$;
  return(N_1, N_2);
end

Figure 19.3: The Kernighan–Lin algorithm

k-exchange is referred to as an *epoch*. The algorithm repeatedly executes such epochs of k-exchanges. If no such k-exchange with positive gain can be found, then the algorithm terminates. The overall algorithm is illustrated in Fig. 19.3.

The Kernighan–Lin algorithm converges rapidly to a local optimum. In fact, a very small number of epochs (fewer than five) may be required for the algorithm to terminate. Of course, there is no guarantee on the required number of epochs, considering that the problem is NP-hard. The running time of each epoch can be amortized to $O(m \cdot \log(n))$ time, where m is the number of edges and n is the number of nodes. Variants of the algorithm have been proposed to speed up the method significantly.

19.3.1.1 Speeding Up Kernighan–Lin

A fast variant of Kernighan–Lin is based on the modifications by Fiduccia and Mattheyses. This version can also handle weights associated with *both* nodes and edges. Furthermore, the approach allows the specification of the level of balance between the two partitions as a ratio. Instead of pairing nodes in an epoch to swap them, one can simply move a single node i from one partition to the other so that the gain D_i of Eq. 19.14 is as large as possible. Only nodes that can move without violating[2] the balancing constraint are considered eligible for a move at each step. After moving node i, it is marked so that it will not be considered again in the current epoch. The values of D_j on the other vertices $j \in N$ are updated to reflect this change. This process is repeated until either all nodes have been considered for a move in an epoch or the balancing criterion prevents further moves. The latter is possible

[2]Moving a node from one partition to the other will frequently cause violations unless some flexibility is allowed in the balancing ratio. In practice, a slight relaxation (or small range) of required balancing ratios may be used to ensure feasible solutions.

when the desired partition ratios are unbalanced, or the nodes do not have unit weights. Note that many potential moves in an epoch might have negative gain. Therefore, as in the original Kernighan–Lin algorithm, only the best partition created during an epoch is made final and the remaining moves are undone. A special data structure was also introduced by Fiduccia and Mattheyses to implement each epoch in $O(m)$ time, where m is the number of edges. In practice, a small number of epochs is usually required for convergence in most real-world networks, although there is no guarantee on the required number of epochs.

While the original improvement of Fiduccia and Mattheyses moves as many vertices as possible in an epoch, it was observed by Karypis and Kumar that it is not necessary to do so. Rather, one can terminate an epoch, if the partitioning objective function does not improve in a predefined number n_p of moves. These n_p moves are then undone, and the epoch terminates. The typical value of n_p chosen is 50. Furthermore, it is not always necessary to move the vertex with the largest possible gain, as long as the gain is positive. Dropping the restriction of finding a vertex with the *largest* gain improves the per-move cost significantly. The improvements from these simple modifications are significant in many scenarios.

19.3.2 Girvan–Newman Algorithm

This algorithm uses edge lengths c_{ij}, rather than the edge weights w_{ij}. The edge lengths may be viewed as in the inverse of the edge weights. In cases, where edge weights are specified, one may heuristically transform them to edge lengths by using $c_{ij} = 1/w_{ij}$, or a suitable application-specific function.

The Girvan–Newman algorithm is based on the intuition that edges with high betweenness have a tendency to connect different clusters. For example, the edges that are incident on the hub nodes in Fig. 19.2 have a high betweenness. Their high betweenness is a result of the large number of pairwise shortest paths between nodes of different communities passing through these edges. Therefore, the disconnection of these edges will result in a set of connected components that corresponds to the natural clusters in the original graph. This disconnection approach forms the basis of the Girvan–Newman algorithm.

The Girvan–Newman algorithm is a top-down hierarchical clustering algorithm that creates clusters by successively removing edges with the highest betweenness until the graph is disconnected into the required number of connected components. Because each edge removal impacts the betweenness values of some of the other edges, the betweenness values of these edges need to be recomputed after each removal. The Girvan–Newman algorithm is illustrated in Fig. 19.4.

The main challenge in the Girvan–Newman algorithm is the computation of the edge betweenness values. The computation of node betweenness values is an intermediary step in the edge-betweenness computation. Recall that all node and edge-betweenness centrality values are defined as a function of the exhaustive set of shortest paths between all source–sink pairs. These betweenness centrality values can, therefore, be decomposed into several additive components, where each component is defined by the subset of the shortest paths originating from a source node s. To compute these betweenness components, a two-step approach is used for each possible source node s:

1. The number of shortest paths from the source node s to every other node is computed.

2. The computations in the first step are used to compute the component $B_s(i)$ of the node betweenness centrality of node i, and the component $b_s(i, j)$ of the edge betweenness centrality of edge (i, j), *that correspond to the subset of shortest paths originating from a particular source node* s.

Algorithm *GirvanNewman*(Graph: $G = (N, A)$, Number of Clusters: k,
Edge lengths: $[c_{ij}]$)
begin
Compute betweenness value of all edges in graph G;
repeat
Remove edge (i, j) from G with highest betweenness;
Recompute betweenness of edges affected by removal of (i, j);
until G has k components remaining;
return connected components of G;
end

Figure 19.4: The Girvan–Newman Algorithm

These source node-specific betweenness centrality components can then be added over all possible source nodes to compute the overall betweenness centrality values.

The first step in the betweenness centrality computation is to create a graph of edges that lie on *at least* one shortest path from node s to some other node. Such edges are referred to as *tight* edges for source node s. The betweenness value component of an edge for a particular source node s can be nonzero only if that edge is tight for that source node. The Dijkstra algorithm, described in Sect. 3.5.1.1 of Chap. 3, is used to determine the shortest path distances $SP(j)$ from the source node s to node j. In order for an edge (i, j) to be tight, the following condition has to hold:

$$SP(j) = SP(i) + c_{ij} \tag{19.16}$$

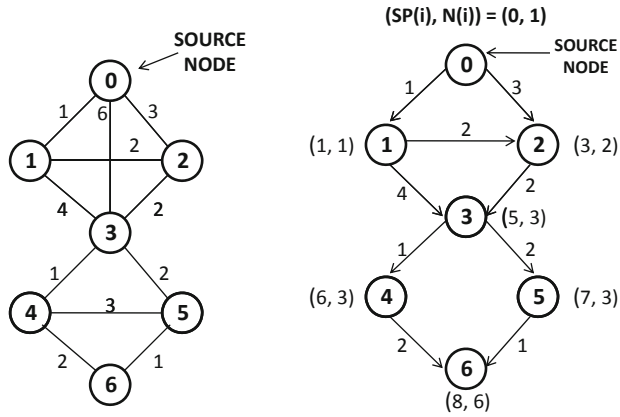
Therefore, the *directed* subgraph $G^s = (N, A^s)$ of tight edges is determined, where $A^s \subseteq A$. The direction of the edge (i, j) is such that $SP(j) > SP(i)$. Therefore, the subgraph of tight edges is a *directed acyclic graph*. An example of a base graph, together with its subgraph of tight edges, is illustrated in Fig. 19.5. The edges are annotated with their lengths. In this case, node 0 is assumed to be the source node. The subgraph of tight edges will obviously vary with the choice of the source node. The shortest-path distances $SP(i)$ of node i from source node 0 are illustrated by the first component of the pair of numbers annotating the nodes in Fig. 19.5b.

The number of shortest paths $N_s(j)$ from the source node s to a given node j is relatively easy to determine from the subgraph of tight edges. This is because the number of paths to a given node is equal to the sum of the number of paths to the nodes incident on it.

$$N_s(j) = \sum_{i:(i,j)\in A^s} N_s(i) \tag{19.17}$$

The algorithm starts by setting $N_s(s) = 1$ for the source node s. Subsequently, the algorithm performs a breadth first search of the subgraph of tight edges, starting with the source node. The number of paths to each node is computed as the sum of the paths to its ancestors in the directed acyclic graph of tight edges, according to Eq. 19.17. The number of shortest paths to each node, from source node 0, is illustrated in Fig. 19.5b by the second component of the pair of numbers annotating each node.

The next step is to compute the component of the betweenness centrality for both nodes and edges starting at the source node s. Let $f_{sk}(i)$ be the fraction of shortest paths between nodes s and k, that pass through node i. Let $F_{sk}(i, j)$ be the fraction of shortest paths



(a) Original graph (b) Tight-edge subgraph

Figure 19.5: Original graph and subgraph of tight edges

between nodes s and k, that pass through edge (i, j). The corresponding components of node betweenness centrality and edge betweenness centrality, specific to node s, are denoted by $B_s(i)$ and $b_s(i, j)$, and they are defined as follows:

$$B_s(i) = \sum_{k \neq s} f_{sk}(i) \tag{19.18}$$

$$b_s(i, j) = \sum_{k \neq s} F_{sk}(i, j) \tag{19.19}$$

It is easy to see that the *unnormalized* values[3] of the node betweenness centrality of i and the edge betweenness centrality of (i, j) may be obtained by respectively summing up each of $B_s(i)$ and $b_s(i, j)$ over the different source nodes s.

The graph G_s of tight edges is used to compute these values. The key is to set up recursive relationships between $B_s(i)$ and $b_s(i, j)$ as follows:

$$B_s(j) = \sum_{i:(i,j) \in A^s} b_s(i, j) \tag{19.20}$$

$$B_s(i) = 1 + \sum_{j:(i,j) \in A^s} b_s(i, j) \tag{19.21}$$

These relationships follow from the fact that shortest paths through a particular node always pass through exactly one of its incoming and outgoing edges, unless they end at that node. The second equation has an additional credit of 1 to account for the paths ending at node i, for which the full fractional credit $f_{si}(i) = 1$ is given to $B_s(i)$.

The source node s is always assigned a betweenness score of $B_s(s) = 0$. The nodes and edges of the directed acyclic tight graph G^s are processed "bottom up," starting at the nodes without any outgoing edges. The score $B_s(i)$ of a node i is finalized, only after the

[3]The normalized values, such as those in Eq. 19.13, may be obtained by dividing the unnormalized values by $n \cdot (n-1)$ for a network with n nodes. The constant of proportionality is irrelevant because the Girvan–Newman algorithm requires only the identification of the edge with the largest betweenness.

scores on all its outgoing edges have been finalized. Similarly, the score $b_s(i, j)$ of an edge (i, j) is finalized only after the score $B_s(j)$ of node j has been finalized. The algorithm starts by setting all nodes j without any outgoing edges to have a score of $B_s(j) = f_{sj}(j) = 1$. This is because such a node j, without outgoing edges, is (trivially) a intermediary between s and j, but it cannot be an intermediary between s and any other node. Then, the algorithm iteratively updates scores of nodes and edges in the bottom-up traversal as follows:

- *Edge Betweenness Update*: Each edge (i, j) is assigned a score $b_s(i, j)$ that is based on partitioning the score $B_s(j)$ into all the incoming edges (i, j) based on Eq. 19.20. The value of $b_s(i, j)$ is proportional to $N_s(i)$ that was computed earlier. Therefore, $b_s(i, j)$ is computed as follows.

$$b_s(i, j) = \frac{N_s(i) \cdot B_s(j)}{\sum_{k:(k,j)\in A^s} N_s(k)} \tag{19.22}$$

- *Node Betweenness Update*: The value of $B_s(i)$ is computed by summing up the values of $b_s(i, j)$ of all its outgoing edges and then adding 1, according to Eq. 19.21.

This entire procedure is repeated over all source nodes, and the values are added up. Note that this provides *unscaled values* of the node and edge betweenness, which may range from 0 to $n \cdot (n-1)$. The (aggregated) value of $B_s(i)$ over all source nodes s can be converted to $C_B(i)$ of Eq. 19.13 by dividing it with $n \cdot (n-1)$.

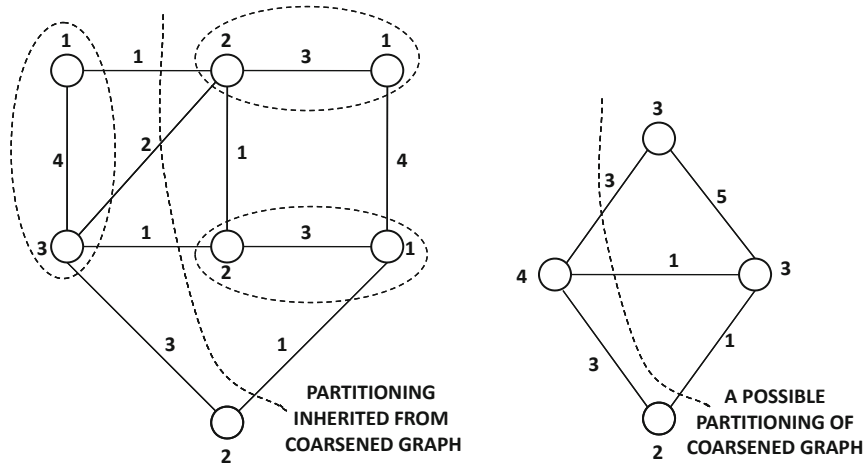
The betweenness values can be computed more efficiently incrementally after edge removals in the Girvan–Newman algorithm. This is because the graphs of tight edges can be computed more efficiently by the use of the incremental shortest path algorithm. The bibliographic notes contain pointers to these methods. Because most of the betweenness computations are incremental, they do not need to be performed from scratch, which makes the algorithm more efficient. However, the algorithm is still quite expensive in practice.

19.3.3 Multilevel Graph Partitioning: METIS

Most of the aforementioned algorithms are quite slow in practice. Even the spectral algorithm, discussed later in this section, is quite slow. The *METIS* algorithm was designed to provide a fast alternative for obtaining high-quality solutions. The *METIS* algorithm allows the specification of weights on *both* the nodes and edges in the clustering process. Therefore, it will be assumed that the weight on each edge (i, j) of the graph $G = (N, A)$ is denoted by w_{ij}, and the weight on node i is denoted by v_i.

The *METIS* algorithm can be used to perform either k-way partitioning or 2-way partitioning. The k-way multilevel graph-partitioning method is based on top-down 2-way recursive bisection of the graph to create k-way partitionings, although variants to perform direct k-way partitioning also exist. Therefore, the following discussion will focus on the 2-way bisection of the graph.

The *METIS* algorithm uses the principle that the partitioning of a *coarsened representation* of a graph can be used to efficiently derive an approximate partition of the original graph. The *coarsened representation* of a graph is obtained by contracting some of the adjacent nodes into a single node. The contraction may result in self-loops that are removed. Such self-loops are also referred to as *collapsed edges*. The weights of the contracted nodes are equal to the sum of the weights of the constituent nodes in the original graph. Similarly, the parallel edges across contracted nodes are consolidated into a single edge with the weights of the constituent edges added together. An example of a coarsened representation of a graph, in which some pairs of adjacent nodes are contracted, is illustrated in Fig. 19.6.



(a) Original graph with inherited partition from coarsened graph

(b) Coarsened graph with partition

Figure 19.6: Illustration of coarsening and partitioning inheritance in uncoarsening

The corresponding node weights and edge weights are also illustrated in the same figure. A good partitioning of this smaller coarsened graph maps to an approximate partitioning of the original graph. Therefore, one possible approach is to compress the original graph into a small one by using a *coarsening heuristic*, then partition this smaller graph more efficiently with any off-the-shelf algorithm, and finally map this partition onto the original graph. An example of mapping a partition on the coarsened graph to the original graph is also illustrated in Fig. 19.6. The resulting partition can be refined with an algorithm, such as the Kernighan–Lin algorithm. The multilevel scheme enhances this basic approach with *multiple levels* of coarsening and refinement to obtain a good trade-off between quality and efficiency. The multilevel partitioning scheme uses three phases:

1. *Coarsening phase:* Carefully chosen sets of nodes in the original graph $G = G_0$ are contracted to create a sequence of successively smaller graphs, $G_0, G_1, G_2 \ldots G_r$. To perform a single step of coarsening from G_{m-1} to G_m, small sets of nonoverlapping and tightly interconnected nodes are identified. Each set of tightly interconnected nodes is contracted into a single node. The heuristics for identifying these node sets will be discussed in detail later. The final graph G_r is typically smaller than a 100 nodes. The small size of this final graph is important in the context of the second partitioning phase. The different levels of coarsening created by this phase create important reference points for a later uncoarsening phase.

2. *Partitioning phase:* Any off-the-shelf algorithm can be used to create a high-quality balanced partitioning from graph G_r. Examples include the spectral approach of Sect. 19.3.4 and the Kernighan–Lin algorithm. It is much easier to obtain a high-quality partitioning with a small graph. This high-quality partitioning provides a good starting point for refinement during the uncoarsening phase. Even relatively poor partitionings of this coarsest graph often map to good partitionings on the uncontracted

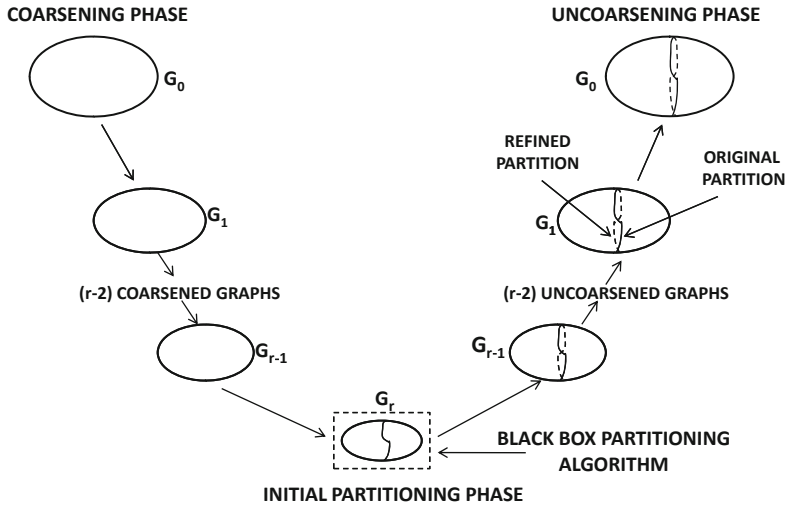


Figure 19.7: The multilevel framework [301] of METIS

graph, because the collapsed edges during coarsening are not eligible to be cut during this phase.

3. *Uncoarsening phase (refinement):* In this phase, the graphs are expanded back to their successively larger versions $G_r, G_{r-1} \ldots G_0$. Whenever the graph G_m is expanded to G_{m-1}, the latter inherits the partitioning from G_m. This inheritance is illustrated in Fig. 19.6. The fast variant of the Kernighan–Lin scheme, discussed in Sect. 19.3.1.1, is applied to this partitioning of G_{m-1} to refine it further before expanding it further to G_{m-2}. Therefore, graph G_{m-2} inherits the refined partition from G_{m-1}. Usually, the refinement phase is extremely fast because the KL-algorithm starts with a very high quality approximate partition of G_{m-1}.

A pictorial representation of the multilevel scheme, based on an illustration in [301], is provided in Fig. 19.7. Note that the second and third phases use off-the-shelf schemes that are discussed in other parts of this chapter. Therefore, the following discussion will focus only on the first phase of coarsening.

A number of techniques are used for coarsening with varying levels of complexity. In the following, a few simple schemes are described that coarsen only by matching *pairs* of nodes in a given phase. In order for a pair of nodes to be matched, they must always be connected with an edge. The coarsened graph will be at least half the size of the original graph in terms of the number of nodes. In spite of the simplicity of these coarsening methods, these schemes turn out to be surprisingly effective in the context of the overall clustering algorithm.

1. *Random edge matching:* A node i is selected at random and matched to an adjacently connected unmatched node that is also selected randomly. If no such unmatched node exists, then the vertex remains unmatched. The matching is performed, until no (adjacent) unmatched pair remains in the graph.

2. *Heavy edge matching:* As in random edge matching, a node i is selected at random and matched to an adjacently connected unmatched node. However, the difference is that the largest weight incident edge (i, j) is used to select the unmatched node j.

The intuition is that it is better to contract heavy edges because they are less likely to be part of an optimal partitioning.

3. *Heavy clique matching:* The contraction of densely connected sets of nodes in the graph will maximize the number of collapsed edges. This method tracks the weight v_i of node i, which corresponds to the number of contracted nodes it represents. Furthermore, the notation s_i denotes the sum of the weights of the collapsed edges at node i (or its precursors) in previous contraction phases. Note that if the contracted node i represents a clique in the original graph, then s_i will approach $v_i \cdot (v_i - 1)/2$. Because it is desirable to contract dense components, one must try to ensure that the value of s_i resulting from the contraction approaches its upper limit. This is achieved by computing the edge density $\mu_{ij} \in (0, 1)$ of edge (i, j):

$$\mu_{ij} = \frac{2 \cdot (s_i + s_j + w_{ij})}{(v_i + v_j) \cdot (v_i + v_j - 1)} \tag{19.23}$$

When nodes across high-density edges are contracted, they typically correspond to cliques in the *original* graph $G = G_0$, if it was unweighted. Even for weighted graphs, the use of high-edge density is generally quite effective. The nodes of the graph are visited in random order. For each node, its highest density unmatched neighbor is selected for matching. Unlike heavy edge matching, the heavy clique matching approach is not myopic to the contractions that have occurred in previous phases of the algorithm.

The multilevel scheme is effective because of its hierarchical approach, where the early clustering of coarsened graphs ensures a good initial global structure to the bisection. In other words, key components of the graph are assigned to the appropriate partitions early on, in the form of coarsened nodes. This partition is then successively improved in refinement phases. Such an approach avoids local optima more effectively because of its "big picture" approach to clustering.

19.3.4 Spectral Clustering

It is assumed that the nodes are unweighted, though the edge (i, j) is associated with the weight w_{ij}. The $n \times n$ matrix of weights is denoted by W. The spectral method uses a graph embedding approach, so that the local clustering structure of the network is preserved by the embedding of the nodes into multidimensional space. The idea is to create a multidimensional representation of the graph so that a standard k-means algorithm can be used on the transformed representation.

The simpler problem of mapping the nodes onto a 1-dimensional space will be discussed first. The generalization to the k-dimensional case is relatively straightforward. We would like to map the nodes in N into a set of 1-dimensional real values $y_1 \ldots y_n$ on a line, so that the distances between these points reflect the connectivity among the nodes. Therefore, it is undesirable for nodes that are connected with high-weight edges to be mapped onto distant points on this line. This can be achieved by determining values of y_i, for which the following objective function O is minimized:

$$O = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (y_i - y_j)^2 \tag{19.24}$$

This objective function penalizes the distances between y_i and y_j with weight proportional to w_{ij}. Therefore, when w_{ij} is very large, the data points y_i and y_j will be more likely to

be closer to one another in the embedded space. The objective function O can be rewritten in terms of the *Laplacian matrix* L of weight matrix W. The Laplacian matrix L is defined as $\Lambda - W$, where Λ is a diagonal matrix satisfying $\Lambda_{ii} = \sum_{j=1}^{n} w_{ij}$. Let the n-dimensional column vector of embedded values be denoted by $\overline{y} = (y_1 \ldots y_n)^T$. It can be shown after some algebraic rearrangement of Eq. 19.24, that the objective function O can be rewritten in terms of the Laplacian matrix:

$$O = 2\overline{y}^T L \overline{y} \tag{19.25}$$

The matrix L is positive semidefinite with nonnegative eigenvalues because the sum-of-squares objective function O is always nonnegative. We need to incorporate a scaling constraint to ensure that the trivial value of $y_i = 0$ for all i is not selected by the optimization solution. A possible scaling constraint is as follows:

$$\overline{y}^T \Lambda \overline{y} = 1 \tag{19.26}$$

The matrix Λ is incorporated in the constraint of Eq. 19.26 to achieve normalization, so that the resulting clusters are more balanced across partitions. If Λ is not used in the constraint, the result is referred to as *unnormalized* spectral clustering. In practice, the effect of this normalization is that low-degree nodes tend to clearly "pick sides" with either large positive or large negative values of y_i, whereas very high-degree nodes, which might also be hub nodes, will be embedded closer to central regions near the origin (see Exercise 7). Note that each diagonal entry Λ_{ii}, which is the sum of the weights of the edges incident on node i, can be viewed as the local density of the network at node i. It can also be shown that incorporating Λ in the constraint approximates an unnormalized embedding in which edge weights $w_{ij} = w_{ji}$ have been divided by the geometric average $\sqrt{\Lambda_{ii} \cdot \Lambda_{jj}}$ of the local densities at their end points (see Exercise 8). As discussed in Chap. 3, normalizing distance or similarity values with local densities is often helpful in obtaining high-quality results that are more accurate in their *local* context.

This constrained optimization formulation can be solved by setting the gradient of its *Lagrangian relaxation* $\overline{y}^T L \overline{y} - \lambda(\overline{y}^T \Lambda \overline{y} - 1)$ to 0. It can be shown that the resulting optimization condition is $\Lambda^{-1} L \overline{y} = \lambda \overline{y}$ where λ is the Lagrangian parameter. In other words, $\overline{y}$ is an eigenvector of $\Lambda^{-1} L$ and λ is an eigenvalue. Furthermore, this optimization condition can be used to easily show that the objective function $O = 2\overline{y}^T L \overline{y}$ evaluates to twice the eigenvalue λ for an eigenvector $\overline{y}$ satisfying this condition. Therefore, among the alternative eigenvector solutions for $\overline{y}$, the optimal solution is the smallest *nontrivial* eigenvector of the *normalized* Laplacian $\Lambda^{-1} L$. The smallest eigenvalue of $\Lambda^{-1} L$ is always 0, and it corresponds to the trivial solution where the node embedding $\overline{y}$ is proportional to $(1, 1, \ldots 1)^T$. Such a trivial 1-dimensional embedding corresponds to mapping every node to the same point. This trivial eigenvector is noninformative. Therefore, it can be discarded, and it is not used in the analysis. The second smallest eigenvector then provides an optimal solution that is more informative.

This model can be generalized to a k-dimensional embedding by setting up an analogous optimization formulation with its decision variables as an $n \times k$ matrix Y with k column vectors $Y = [\overline{y_1} \ldots \overline{y_k}]$ representing each dimension of the embedding. This optimization formulation minimizes the trace of the $k \times k$ matrix $Y^T L Y$ subject to the normalization constraints $Y^T \Lambda Y = I$. Because of the presence of Λ in the constraint, the columns of Y will not necessarily be orthogonal. The optimal solutions for these k column vectors can be shown to be proportional to the successive directions corresponding to the (not necessarily orthogonal) right eigenvectors of the asymmetric matrix $\Lambda^{-1} L$ with increasing eigenvalues. After discarding the first trivial eigenvector $\overline{e_1}$ with eigenvalue $\lambda_1 = 0$, this results in a set

of k eigenvectors $\overline{e_2}, \overline{e_3} \ldots \overline{e_{k+1}}$, with corresponding eigenvalues $\lambda_2 \leq \lambda_3 \ldots \leq \lambda_{k+1}$. Because k eigenvectors were selected, this approach creates an $n \times k$ matrix $D_k = Y$, corresponding to a k-dimensional embedding of each of the n nodes. Note that even though the normalization constraint $Y^T \Lambda Y = I$ will not result in columns of D_k having an L_2-norm of 1, each column (eigenvector) of D_k is scaled to an L_2-norm of 1 as a post-processing[4] step. Because of this column scaling, the $n \times k$ matrix D_k does not exactly reflect the original optimization formulation in terms of Y. The resulting k-dimensional embedding preserves the clustering structure of the nodes because the optimization formulation of Y tries to minimize distances between highly connected nodes. Therefore, any multidimensional clustering algorithm discussed in Chap. 6, such as k-means, can be applied to this embedded representation to generate the clusters on the nodes. This formulation is also sometimes referred to as the *random walk version* of spectral clustering because of an interpretation in terms of random walks. It is noteworthy that the small eigenvectors of the normalized Laplacian $\Lambda^{-1}L$ are the same as the large eigenvectors of the stochastic transition matrix $\Lambda^{-1}W$ (see Exercise 15).

An *equivalent* way of setting up the spectral clustering model is to use the related vector of decision variables $\overline{z} = \sqrt{\Lambda}\overline{y}$ in the optimization formulation of Eqs. 19.25 and 19.26. This related version is referred to as the *symmetric version* of the spectral clustering model, although it is different from the random walk version only in terms of the scaling of decision variables. By setting $\overline{z} = \sqrt{\Lambda}\overline{y}$, it can be shown that the earlier formulation is equivalent to optimizing $\overline{z}^T \Lambda^{-1/2} L \Lambda^{-1/2} \overline{z}$ subject to $\overline{z}^T \overline{z} = 1$. We determine the smallest k (orthogonal) eigenvectors of the *symmetric normalized* Laplacian $\Lambda^{-1/2} L \Lambda^{-1/2}$, excluding the first. Each eigenvector of this matrix can also be (proportionally) obtained by pre-multiplying the aforementioned solution Y of the random walk formulation with the diagonal matrix $\sqrt{\Lambda}$. This relationship also reflects the relationship between $\overline{z}$ and $\overline{y}$. The eigenvalues are the same in both cases. For example, the first eigenvector with eigenvalue 0 will no longer be a vector of 1s, but the various entries will be proportional to $(\sqrt{\Lambda_{11}} \ldots \sqrt{\Lambda_{nn}})^T$. Because of this differential scaling of various nodes, high-degree nodes will tend to have larger (absolute) coordinate values in the symmetric version. By selecting the smallest k eigenvectors, one can generate an $n \times k$ multidimensional representation D_k of the entire set of n nodes. Just as the random walk version scales each column of D_k to unit norm in the final step, the symmetric version scales each row of D_k to unit norm. The final step of row scaling is a heuristic enhancement to adjust for the differential scaling of nodes with various degrees, and it is not a part of the optimization formulation. Interestingly, even if the rows of the random walk solution Y had been scaled to unit norm (instead of scaling the columns to unit norm), exactly the same solution would be obtained as that obtained by scaling the rows of the symmetric solution Z to unit norm (see Exercise 13).

Although the two different ways of performing spectral clustering are equivalent in terms of the optimization problem solved, there are differences in terms of the heuristic scaling adjustments. The scaling relationships are illustrated in Fig. 19.8. It is evident from Fig. 19.8 that the main *practical* difference between the two methods is regulated only by the *heuristic* scaling used in the final phase, rather than their respective optimization models. Because of the scaling variations, the clusters obtained in the two cases will not be exactly the same. The relative quality will depend on the data set at hand. These optimization problems can also be understood as linear programming relaxations of integer-programming formulations of balanced minimum cut problems. However, the minimum-cut explanation does not

[4]In practice, the *unit* eigenvectors of $\Lambda^{-1}L$ can be directly computed, and therefore an explicit post-processing step is not required.

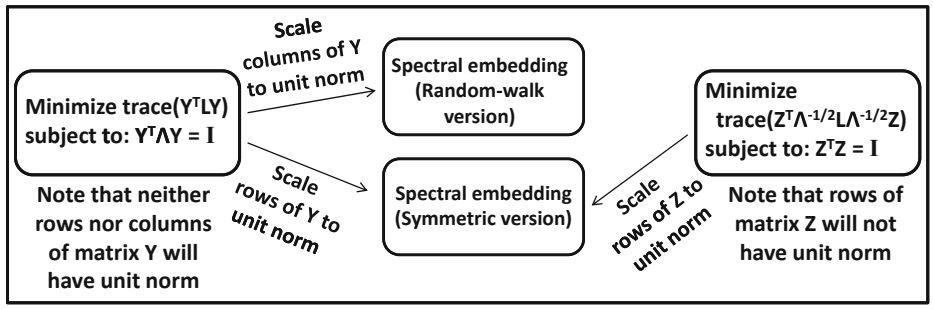


Figure 19.8: Scaling relationships between random walk and symmetric versions of spectral clustering

intuitively generalize to the relaxed version of the problem because eigenvectors have both positive and negative components.

19.3.4.1 Important Observations and Intuitions

A few observations are noteworthy about the relationships between spectral clustering, *PageRank*, and eigenvector analysis:

1. *Normalized random walk Laplacian:* The smallest *right* eigenvectors of $\Lambda^{-1}L = \Lambda^{-1}(\Lambda - W) = I - P$ are used for the random walk embedding, where P is the stochastic transition matrix of the graph. The smallest right eigenvectors of $I - P$ are the same as the largest right eigenvectors of P. The largest right eigenvector of P has eigenvalue 1. It is noteworthy that the largest *left* eigenvector of P, which also has eigenvalue 1, yields the *PageRank* of the graph. Therefore, both the left and the right eigenvectors of the stochastic transition matrix P yield different insights about the network.

2. *Normalized symmetric Laplacian:* The smallest eigenvectors of the symmetric Laplacian $\Lambda^{-1/2}(\Lambda - W)\Lambda^{-1/2}$ are the same as the largest eigenvectors of the symmetric matrix $\Lambda^{-1/2}W\Lambda^{-1/2}$. The matrix $\Lambda^{-1/2}W\Lambda^{-1/2}$ can be viewed as a normalized and sparsified similarity matrix of the graph. Most forms of nonlinear embeddings such as *SVD*, *Kernel PCA*, and *ISOMAP* are extracted as large eigenvectors of similarity matrices (cf. Table 2.3 of Chap. 2). It is the choice of the similarity matrix that regulates the varying properties of these different embeddings.

3. *Goal of normalization:* Spectral clustering is more effective when the unnormalized Laplacian L is normalized with the node-degree matrix Λ. While it is possible to explain this behavior with a cut-interpretation of spectral clustering, the intuition does not generalize easily to continuous embeddings with both positive and negative eigenvector components. A simpler way of understanding normalization is by examining the similarity matrix $\Lambda^{-1/2}W\Lambda^{-1/2}$ whose large eigenvectors yield the normalized spectral embedding. In this matrix, the edge similarities are normalized by the geometric mean of the node degrees at their end points. This can be viewed as a normalization of the edge similarities with a local measure of the network density. As discussed in Chap. 3, normalizing similarity and distance functions with local density is helpful even in the case of multidimensional data mining applications. One of the most well-known algorithms for outlier analysis in multidimensional data, referred to as *LOF*,

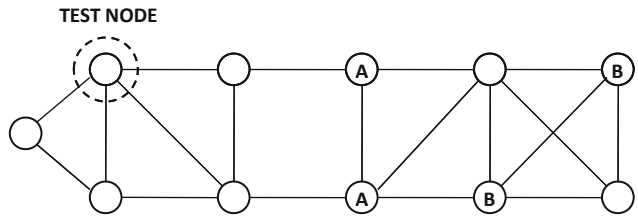


Figure 19.9: Label sparsity issues in collective classification

also uses this principle. Normalization will yield more balanced clusters in networks with widely varying density over the network.

19.4 Collective Classification

In many social networking applications, labels may be associated with nodes. For example, consider the case of a social networking application, where it is desirable to determine all individuals interested in *golf*. The labels of a small number of actors may already be available. It is desirable to use the available labels to perform the classification of nodes for which the label is not known.

The solution to this model is crucially dependent on the notion of homophily. Because nodes with similar properties are usually connected, it is reasonable to assume that this is also true of node labels. A simple solution to this problem is to examine the k labeled nodes in the proximity of a given node and report the majority label. This approach is, in fact, the network analog of a nearest neighbor classifier. However, such an approach is generally not possible in collective classification because of the sparsity of node labels. An example of a network is illustrated in Fig. 19.9, in which the two classes are labeled A and B. The remaining nodes are unlabeled. For the test node in Fig. 19.9, it is evident that it is generally closer to instances of A in the network structure, but there is no labeled node *directly* connected to the test instance.

Thus, it is evident that one must not only use the direct connections to labeled nodes, but also use the indirect connections through unlabeled nodes. Thus, collective classification in networks are always performed in a *transductive semisupervised setting*, where the test instances and training instances are classified jointly. In fact, as discussed in Sect. 11.6.3 of Chap. 11, collective classification methods can be used for semisupervised classification of any data type by transforming the data into a similarity graph. *Thus, the collective classification problem is important not only from the perspective of social network analysis, but also for semisupervised classification of any data type.*

19.4.1 Iterative Classification Algorithm

The *Iterative Classification Algorithm (ICA)* is one of the earliest classification algorithms in the literature and has been applied to a wide variety of data domains. The algorithm has the capability to use content associated with the nodes for classification. This is important because many social networks have text content associated with the nodes in the form of user posts. Furthermore, in cases where this framework is used[5] for semisupervised classification

[5] cf. Sect. 11.6.3 of Chap. 11.

Algorithm *ICA*(Graph $G = (N, A)$, Weights: $[w_{ij}]$, Node Class Labels: $\mathcal{C}$,
Base Classifier: $\mathcal{A}$, Number of Iterations: T)
begin
repeat
Extract link features at each node with current training data;
Train classifier $\mathcal{A}$ using both link and content features of
current training data and predict labels of test nodes;
Make (predicted) labels of most "certain" n_t/T
test nodes final, and add these nodes to training
data, while removing them from test data;
until T iterations;
end

Figure 19.10: The iterative classification algorithm (ICA)

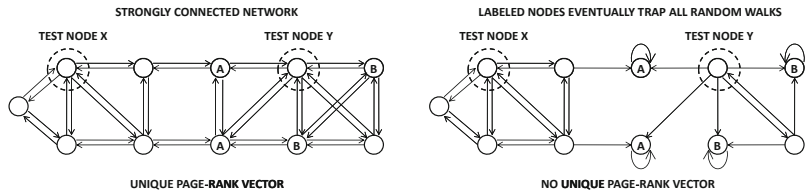
of relational data with similarity graphs, the relational features continue to be available at the nodes for more effective classification.

Consider the (undirected) network $G = (N, A)$ with class labels are drawn from $\{1 \ldots k\}$. Each edge $(i, j) \in A$ is associated with the weight w_{ij}. Furthermore, the content $\overline{X_i}$ is available at the node i in the form of a multidimensional feature vector. The total number of nodes is denoted by n, from which n_t nodes are unlabeled test nodes.

An important step of the *ICA* algorithm is to derive a set of *link features* in addition to the available content features in $\overline{X_i}$. The most important link features correspond to the distribution of the classes in the immediate neighborhood of the node. Therefore a feature is generated for each class, containing the fraction of its incident nodes belonging to that class. For each node i, its adjacent node j is weighted by w_{ij} for computing its credit to the relevant class. It is also possible, in principle, to derive other link features based on structural properties of the graph such as the degree of the node, *PageRank* values, number of closed triangles involving the node, or connectivity features. Such link features can be derived on the basis of an application-specific understanding of the network data set.

The basic ICA is structured as a meta-algorithm. A base classifier $\mathcal{A}$ is leveraged within an iterative framework. Many different base classifiers have been used in different implementations, such as the naive Bayes classifier, logistic regression classifier, and a neighborhood voting classifier. The main requirement is that these classifiers should be able to output a numeric score that quantifies the likelihood of a node belonging to a particular class. While the framework is independent of specific choice of classifier, the use of the naive Bayes classifier is particularly common because of the interpretation of its numeric score as a probability. Therefore, the following discussion will assume that the algorithm $\mathcal{A}$ is instantiated to the naive Bayes classifier.

The link and content features are used to train the naive Bayes classifier. For many nodes, it is difficult to robustly estimate important class-specific features, such as the fractional presence of the different classes in their neighborhood. This is a direct result of label sparsity, and it makes the class predictions of such nodes unreliable. Therefore, an iterative approach is used for augmenting the training data set. In each iteration, n_t/T (test) node labels are made "certain" by the approach, where T is a user-defined parameter controlling the maximum number of iterations. The test nodes, for which the Bayes classifier exhibits the highest class membership probabilities, are selected to be made final. These labeled test



(a) No absorbing state (b) With absorbing states

Figure 19.11: Creating directed transition graphs from undirected graph of Fig. 19.9

nodes can then be added to the training data, and the classifier is retrained by extracting the link features again with the augmented training data set. The approach is repeated until the labels of all nodes have been made final. Because the labels of n_t/T nodes are finalized in each iteration, the entire process terminates in exactly T iterations. The overall pseudocode is illustrated in Fig. 19.10.

One advantage of the ICA is that it can seamlessly use content and structure in the classification process. The classifier can automatically select the most relevant features using off-the-shelf feature selection algorithms discussed in Chap. 10. This approach also has the merit that it is not strongly dependent on the notion of homophily, and can, therefore, be used for domains beyond social network analysis. Consider an adversarial relationship network in which nodes connected by links might have different labels. In such cases, the *ICA* algorithm will automatically learn the correct importance of adjacent class distributions, and therefore it will yield accurate results. This property is not true of most of the other collective classification methods, which are *explicitly* dependent on the notion of homophily. On the other hand, the errors made in the earlier phases of iterative classification can propagate and multiply in later phases because of augmented training examples with incorrect labels. This can increase the cumulative error in noisy training data sets.

19.4.2 Label Propagation with Random Walks

The label propagation method directly uses random walks on the undirected network structure $G = (N, A)$. The weight of edge (i, j) is denoted by $w_{ij} = w_{ji}$. To classify an unlabeled node i, a random walk is executed starting at node i and terminated at the first labeled node encountered. The class at which the random walk has the highest probability of termination is reported as the predicted label of node i. The intuition for this approach is that the walk is more likely to terminate at labeled nodes in the proximity of node i. Therefore, when many nodes of a particular class are located in its proximity, then the node i is more likely to be labeled with that class.

An important assumption is that the graph needs to be *label connected*. In other words, every unlabeled node needs to be able to reach a labeled node in the random walk. For undirected graphs $G = (N, A)$, this means that every connected component of the graph needs to contain at least one labeled node. In the following discussion, it will be assumed that the graph $G = (N, A)$ is undirected and label-connected.

The first step is to model the random walks in such a way that they always terminate at their *first arrival* at labeled nodes. This can be achieved by removing outgoing edges from labeled nodes and replacing them with self-loops. Furthermore, to use a random walk approach, we need to convert the undirected graph $G = (N, A)$ into a directed graph $G' = (N, A')$ with an $n \times n$ transition matrix $P = [p_{ij}]$:

1. For each undirected edge $(i, j) \in A$, directed edges (i, j) and (j, i) are added to A' between the corresponding nodes. The transition probability p_{ij} of edge (i, j) is defined as follows:

$$p_{ij} = \frac{w_{ij}}{\sum_{k=1}^{n} w_{ik}} \tag{19.27}$$

The transition probability p_{ji} of edge (j, i) is defined as follows:

$$p_{ji} = \frac{w_{ji}}{\sum_{k=1}^{n} w_{jk}} \tag{19.28}$$

For example, the directed transition graph created from the undirected graph of Fig. 19.9 is illustrated in Fig. 19.11a.

2. All outgoing edges from labeled nodes are removed from the graph G' constructed in the previous step and replaced with a self-loop of transition probability 1. Such nodes are referred to as *absorbing* nodes because they trap the random walk after an incoming transition. An example of the final transition graph is illustrated in Fig. 19.11b. Therefore, for each absorbing node i, the ith row of P is replaced with the ith row of the identity matrix.

Assume that the final $n \times n$ transition matrix is denoted by $P = [p_{ij}]$. For any absorbing node i, the value of p_{ik} is 1 only when $i = k$, and 0 otherwise. The transition matrix P does *not* have a unique steady-state probability distribution (or, *PageRank* vector), because of the presence of absorbing[6] components. The steady-state probability distribution *is dependent on the starting state of the random walk.* For example, a random walk starting at test node X in Fig. 19.11b will always eventually end at label A, whereas a walk starting with node Y might end at either label A or B. It is noteworthy that the *PageRank* computation of Sect. 18.4.1 in Chap. 18 ensures unique steady-state probabilities by using teleportation to implicitly create a strongly connected transition graph. Interestingly, the modifications that create absorbing nodes have exactly the opposite effect because the steady state probability distribution depends on the starting state. Strong connectivity of the transition graph is required to ensure a unique steady-state distribution. However, if the starting state is fixed, then each node does have a steady state probability distribution.

For any given starting node i, the steady-state probability distribution has positive values only at labeled nodes. This is because a random walk will eventually reach an absorbing node in a label-connected graph, and it will never emerge from that node. Therefore, if one can estimate the steady-state probability distribution for starting node i, then the probability values of the labeled nodes in each class can be aggregated. The class with the highest probability is reported as the relevant label of the node i.

How can the steady-state probability be computed for a particular starting node i? Let $\overline{\pi}^{(t)}$ represent the n-dimensional (row) probability vector after t steps, starting with a particular initial state $\overline{\pi}^{(0)}$. When the starting state is node i, the value of $\overline{\pi}^{(0)}$ is 1 for the ith component in this vector, and 0 otherwise. Then, we have:

$$\overline{\pi}^{(t)} = \overline{\pi}^{(t-1)} P \tag{19.29}$$

[6] In other words, the underlying Markov chain is not strongly connected, and therefore not ergodic. See the description of the *PageRank* algorithm in Chap. 18.

By recursively applying the aforementioned condition t times, and then setting $t = \infty$, it is possible to show the following:

$$\overline{\pi}^{(t)} = \overline{\pi}^{(0)} P^t \tag{19.30}$$

$$\overline{\pi}^{(\infty)} = \overline{\pi}^{(0)} P^\infty \tag{19.31}$$

How can the steady-state transition matrix P^∞ be computed? A key observation is that the largest magnitude of the eigenvalue of a stochastic matrix is always 1 (see Exercise 7 of Chap. 18). Therefore, P may be expressed as follows:

$$P = V \Delta V^{-1} \tag{19.32}$$

Here, V is an $n \times n$ matrix, whose columns contain the eigenvectors, and Δ is a diagonal matrix containing the eigenvalues, all of which have magnitude no larger than 1. Note that stochastic matrices with absorbing components will have an eigenvector with unit eigenvalue for each absorbing component. Then, by multiplying P with itself $(t-1)$ times, we get:

$$P^t = V \Delta^t V^{-1} \tag{19.33}$$

In the limit where t approaches infinity, Δ^t will contain diagonal values of only 0 or 1. Any eigenvalue in the original matrix Δ with magnitude less than 1 will approach 0 in Δ^∞. In other words, Δ^∞ can be computed easily from Δ. Therefore, if V has been computed, then P^∞ can be computed easily as well. A further optimization is that the steady-state transition matrix P^∞ can be efficiently computed by determining only the l leading eigenvectors of P, where l is the number of labeled (absorbing) nodes. Refer to the bibliographic notes for more details of this optimization.

After P^∞ has been computed, it is relatively straightforward to compute the n-dimensional node probability vector $\overline{\pi}^{(\infty)}$ that results from starting the random walk at node i. When the starting state is (unlabeled) node i, the n-dimensional vector for the starting state $\overline{\pi}^{(0)}$ contains 1 in the ith component, and 0 otherwise. According to our earlier discussion, one can compute $\overline{\pi}^{(\infty)} = \overline{\pi}^{(0)} P^\infty$. Note that $\overline{\pi}^{(\infty)}$ will contain positive probabilities only for the labeled nodes, which are also absorbing states. By summing up the probabilities in $\overline{\pi}^{(\infty)}$ of labeled nodes belonging to each class, one can obtain the probability of each class for unlabeled node i. The class with the maximum probability is reported as the relevant label.

There is, however, a simpler way of computing the class probability distributions of all unlabeled nodes in one shot, rather than having to explicitly compute P^∞, and then trying different starting vectors for $\overline{\pi}^{(0)}$. For each class $c \in \{1 \ldots k\}$, let $N_c \subseteq N$ be the set of labeled nodes belonging to that class. In order for unlabeled node i to belong to class c, a walk starting at node i must end at a node in N_c. The probability of this event is given by $\sum_{j \in N_c} [P^\infty]_{ij}$. Let $\overline{Y_c}$ be a column vector with n entries such that the jth entry is 1, if node j belongs to class c, and 0, otherwise. Then, it is easy to see that the ith entry of the column vector $\overline{Z_c} = P^\infty \overline{Y_c}$ is equivalent to $\sum_{j \in N_c} [P^\infty]_{ij}$, which is the sum of the probabilities of a walk starting at unlabeled node i terminating at various nodes belonging to class c.

Therefore, we need to compute $\overline{Z_c}$ for each class $c \in \{1 \ldots k\}$. Let Y be an $n \times k$ matrix for which the cth column is $\overline{Y_c}$. Similarly, let Z be an $n \times k$ matrix for which the cth column is $\overline{Z_c}$. Then Z can be obtained with simple matrix multiplication between P^∞ and Y.

$$Z = P^\infty Y \tag{19.34}$$

The class with the maximum probability in Z for unlabeled node (row) i may be reported as its class label. This approach is also referred to as the *rendezvous approach* to label propagation.

We make a few important observations. If the ith row of P is absorbing then it is the same as the ith row of the identity matrix. Therefore, premultiplying Y with P for any number of times will not change the ith row of Y. In other words, rows of Z that correspond to labeled nodes will be fixed to the corresponding rows of Y. Therefore, predictions of labeled nodes are fixed to their training labels. For unlabeled nodes, the rows of Z will always sum to 1 in label-connected networks. This is because the sum of the values in row i in Z is equal to the probability that a random walk starting at node i reaches an absorbing state. In label-connected networks, every random walk will eventually reach an absorbing state.

19.4.2.1 Iterative Label Propagation: The Spectral Interpretation

Equation 19.34 suggests a simple iterative approach for computing the label probabilities in Z, rather than computing P^{∞}. One can initialize $Z^{(0)} = Y$ and then repeatedly use the following update for increasing value of iteration index t.

$$Z^{(t+1)} = PZ^{(t)} \tag{19.35}$$

It is easy to see that $Z^{(\infty)}$ is the same as the value of Z in Eq. 19.34. For labeled (absorbing) node i, the ith row of Z will always be unaffected by the update because the ith row of P is the same as that of the identity matrix. The label-propagation update is executed to convergence. In practice, a relatively small number of iterations are required to reach convergence.

The label propagation update can be rearranged to show that the final solution Z will satisfy the following relationship at convergence:

$$(I - P)Z = 0 \tag{19.36}$$

Note that $I-P$ is simply the normalized (random walk) Laplacian of the adjacency matrix of the network G' with absorbing states. Furthermore, each column of Z is a eigenvector of this Laplacian with eigenvalue 0. In unsupervised spectral clustering, the first eigenvector with eigenvalue 0 is discarded because it is not informative. However, in collective classification, there are additional eigenvectors of $(I - P)$ with eigenvalue 0 because of the presence of absorbing states. Each class-specific column of Z contains a different eigenvector with eigenvalue 0. In fact, the label propagation solution can also be derived with an optimization formulation similar to spectral clustering on the original undirected graph G. In this case, the optimization formulation uses a similar objective function as spectral clustering *with the additional constraint* that the embedded values of all labeled nodes are fixed to 1 for a particular (say, the cth) class and they are fixed to 0 for the remaining classes. The embedded values of only the unlabeled nodes are unconstrained decision variables. The solution to each such optimization problem can be shown to be an eigenvector of $(I - P)$, with eigenvalue 0. Iterative label propagation converges to these eigenvectors.

19.4.3 Supervised Spectral Methods

Spectral methods can be used in two different ways for collective classification of graphs. The first method directly transforms the graph to multidimensional data to facilitate the use of a multidimensional classifier such as a k-nearest neighbor classifier. The embedding

approach is identical to that used in spectral clustering except that the class information is incorporated within the embedding. The second method directly learns an $n \times k$ class probability matrix Z with an optimization formulation related to spectral clustering. This class probability matrix Z is similar to that derived in label propagation. Interestingly, the second method is also closely related to label propagation.

19.4.3.1 Supervised Feature Generation with Spectral Embedding

Let $G = (N, A)$ be the undirected graph with weight matrix W. The approach consists of the following steps, the first of which is to augment G with class-based supervision:

1. Add an edge with weight μ between each pair of nodes with the same label in G. If an edge already exists between a pair of such nodes, then the two edges are consolidated by adding μ to the weight of the existing edge. The resulting graph is denoted by G^+. The parameter μ controls the level of supervision from existing labels.
2. Use the spectral embedding approach of Sect. 19.3.4 to generate an r-dimensional embedding of the augmented graph G^+.
3. Apply any multidimensional classifier, such as a nearest neighbor classifier, on the embedded data.

The value of μ may be tuned with the use of cross-validation. Note that this approach does not directly learn the class probabilities. Rather, it creates a feature representation that implicitly incorporates both the homophily effects and the existing label information. This feature representation is sensitive to both network locality and label distribution. Therefore, it can be used to design an effective multidimensional classifier.

19.4.3.2 Graph Regularization Approach

The graph regularization approach learns the labels of the nodes directly with an optimization formulation related to spectral clustering. let Z be an $n \times k$ matrix of optimization variables, in which the (i, c)th entry denotes the propensity of node i to belong to label c. When the (i, c)th entry is large, it indicates that node i is more likely to belong to label c. Therefore, for the ith row of Z, the index of the largest of the k entries provides a prediction of the class label of node i. The column-vector $\overline{Z_c}$ denotes the cth column of Z for $c \in \{1 \ldots k\}$. Furthermore, Y is an $n \times k$ binary matrix containing the label information. If the ith node is labeled, then exactly one entry in the ith row of Y is 1, corresponding to the relevant class label. Other entries are 0. For unlabeled nodes, all entries in the corresponding row of Y are 0. The cth column of Y is denoted by the column vector $\overline{Y_c}$.

This approach directly uses the weighted matrix W of an undirected graph $G = (N, A)$ (e.g., Fig. 19.9) rather than a directed transition graph. The variables in the matrix Z are derived with an optimization formulation related to spectral clustering. Each n-dimensional vector $\overline{Z_c}$ is viewed as a 1-dimensional embedding of the n nodes. The goal of this optimization formulation is two-fold, which is reflected in the two additive terms of the objective function:

1. *Smoothness (homophily) objective:* For each class $c \in \{1 \ldots k\}$, the nodes connected with high-weight edges should be mapped to similar values in $\overline{Z_c}$. This goal is identical to the unsupervised objective function in spectral clustering. In this case, the *symmetric* Laplacian L^s is used because of its better convergence properties:

$$L^s = I - \Lambda^{-1/2} W \Lambda^{-1/2} \tag{19.37}$$

Here, Λ is a diagonal matrix in which the ith diagonal entry contains the sum of the ith row entries of the $n \times n$ weight matrix W. For brevity, we denote the normalized weight matrix by $S = \Lambda^{-1/2} W \Lambda^{-1/2}$. Therefore, the smoothness term O_s in the objective function may be written as follows:

$$O_s = \sum_{c=1}^{k} \overline{Z_c}^T L^s \overline{Z_c} = \sum_{c=1}^{k} \overline{Z_c}^T (I - S) \overline{Z_c} \tag{19.38}$$

This term is referred to as the *smoothness* term because it ensures that the *predicted* label propensities Z vary smoothly along edges, especially if the weights of the edges are large. This term can also be viewed as a *local consistency* term.

2. *Label-fitting objective:* Because the embedding Z is designed to mimic Y as closely as possible the value of $||\overline{Z_c} - \overline{Y_c}||^2$ should be as small as possible for each class c. Note that unlabeled nodes are included within $||\overline{Z_c} - \overline{Y_c}||^2$, and for those nodes, this term serves as a *regularizer*. The goal of a *regularizer* is to avoid[7] ill-conditioned solutions and overfitting in optimization models.

$$O_f = \sum_{c=1}^{k} ||\overline{Y_c} - \overline{Z_c}||^2 \tag{19.39}$$

This term can also be viewed as a *global consistency* term.

The overall objective function may be constructed as $O = O_s + \mu O_f$, where μ defines the weight of the label-fitting term. The parameter μ reflects the trade-off between the two criteria. Therefore, the overall objective function may be written as follows:

$$O = \sum_{c=1}^{k} \overline{Z_c}^T (I - S) \overline{Z_c} + \mu \sum_{c=1}^{k} ||\overline{Y_c} - \overline{Z_c}||^2 \tag{19.40}$$

To optimize this objective function, one must use the partial derivative with respect to the different decision variables in $\overline{Z_c}$ and set it to zero. This yields the following condition:

$$(I - S)\overline{Z_c} + \mu(\overline{Z_c} - \overline{Y_c}) = 0 \quad \forall c \in \{1 \ldots k\} \tag{19.41}$$

Because this condition is true for each class $c \in \{1 \ldots k\}$ one can write the aforementioned condition in matrix form as well:

$$(I - S)Z + \mu(Z - Y) = 0 \tag{19.42}$$

One can rearrange this optimization condition as follows:

$$Z = \frac{SZ}{1 + \mu} + \frac{\mu}{1 + \mu} Y \tag{19.43}$$

The goal is to determine a solution Z of this optimization condition. This can be achieved iteratively by initializing $Z^{(0)} = Y$ and then iteratively updating $Z^{(t+1)}$ from $Z^{(t)}$ as follows:

$$Z^{(t+1)} = \frac{SZ^{(t)}}{1 + \mu} + \frac{\mu}{1 + \mu} Y \tag{19.44}$$

[7] In this case, the regularizer ensures that no single entry in $\overline{Z_c}$ for unlabeled nodes is excessively large.

This solution is iterated to convergence. It can be shown that the approach converges to the following solution:

$$Z^{(\infty)} = \frac{\mu}{1+\mu}\left(I + \frac{S}{1+\mu} + \left(\frac{S}{1+\mu}\right)^2 + \ldots\right) Y = \frac{\mu}{1+\mu}\left(I - \frac{S}{1+\mu}\right)^{-1} Y \tag{19.45}$$

Intuitively, the matrix $\left(I - \frac{S}{1+\mu}\right)^{-1} = \left(I + \frac{S}{1+\mu} + \left(\frac{S}{1+\mu}\right)^2 + \ldots\right)$ is an $n \times n$ matrix of pairwise *weighted Katz coefficients* (cf. Definition 19.5.4) between nodes. In other words, the propensity of node i to belong to class j is predicted as a sum of its weighted Katz coefficients with respect to labeled nodes of class j. Because the Katz measure predicts links (cf. Sect. 19.5) between nodes, this approach illustrates the connections between collective classification and link prediction.

It is possible to learn the optimal value of μ with the use of cross-validation. It is noteworthy that, unlike the aforementioned label propagation algorithm with absorbing states, this approach only *biases* Z with the labels, and it does not *constrain* the rows in Z to be the same as the corresponding rows of Y for labeled nodes. In fact, the matrix Z can provide a label prediction for an already labeled node that is different from its original training label. Such cases are likely to occur when the original labeling in the training data is error-prone and noisy. The regularization approach is, therefore, more flexible and robust for networks containing noisy and error-prone training labels.

19.4.3.3 Connections with Random Walk Methods

Even though the graph regularization approach is derived using spectral methods, it is also related to random walk methods. The $n \times k$ matrix-based update Eq. 19.44 can be decomposed into k different vector-based update equations, one for each n-dimensional column $\overline{Z_c}$ of Z:

$$\overline{Z_c} = \frac{S\overline{Z_c}}{1+\mu} + \frac{\mu}{1+\mu}\overline{Y_c} \quad \forall c \in \{1 \ldots k\} \tag{19.46}$$

Each of these update equations is algebraically similar to a personalized *PageRank* equation where S replaces the transition matrix and the restart probability is $\frac{\mu}{1+\mu}$ at labeled nodes belonging to a particular class c. The vector $\overline{Y_c}$ is analogous to the personalized restart vector for class c multiplied with the number of training nodes in class c. Similarly, the vector $\overline{Z_c}$ is analogous to the personalized *PageRank* vector of class c multiplied with the number of training nodes in class c. Therefore, the class-specific Eq. 19.46 can be viewed as a personalized *PageRank* equation, scaled in proportion to the prior probability of class c. Of course, the symmetric matrix S is not truly a stochastic transition matrix because its columns do not sum to 1. Therefore, the results cannot formally be viewed as personalized *PageRank* probabilities.

Nevertheless, this algebraic similarity to personalized *PageRank* suggests the possibility of a closely related family of random walk methods, similar to label propagation. For example, instead of using a nonstochastic matrix S derived from spectral clustering, one might use a stochastic transition matrix P. In other words, Eqs. 19.27 and 19.28 are used to derive $P = \Lambda^{-1}W$. However, one difference from the transition matrix P used in label-propagation methods is that the network structure is not altered to create absorbing states. In other words, the directed transition graph of Fig. 19.11a is used, rather than that of Fig. 19.11b to derive P. Replacing S with P in Eq. 19.46 leads to a variant of the label propagation

update (cf. Eq. 19.35) in which labeled nodes are no longer constrained to be predicted to their original label.

Replacing S with P^T in Eq. 19.46 leads to the (class-prior scaled) personalized *PageRank* equations. This is equivalent to executing the personalized *PageRank* algorithm k times, where the personalization vector for the cth execution restarts at labeled nodes belonging to the cth class. Each class-specific personalized *PageRank* probability is multiplied with the prior probability of that class, or, equivalently, the number of labeled training nodes in that class. For each node, the class index that yields the highest (prior-scaled) personalized *PageRank* probability is reported. The performance of these alternative methods is dependent on the data set at hand.

19.5 Link Prediction

In many social networks, it is desirable to predict future links between pairs of nodes in the network. For example, commercial social networks, such as Facebook, often recommend users as potential friends. In general, both structure and content similarity may be used to predict links between pairs of nodes. These criteria are discussed below:

- *Structural measures:* Structural measures typically use the principle of *triadic closure* to make predictions. The idea is that two nodes that share similar nodes in their neighborhoods are more likely to become connected in the future, if they are not already connected.

- *Content-based measures:* In these cases, the principle of homophily is used to make predictions. The idea is that nodes that have similar content are more likely to become linked. For example, in a bibliographic network containing scientific co-author relations, a node containing the keyword "data mining" is more likely to be connected to another node containing the keyword "machine learning."

While content-based measures have been shown to have potential in enhancing link prediction, the results are rather sensitive to the network at hand. For example, in a network such as Twitter, where the content is the form of short and noisy tweets with many nonstandard acronyms, content-based measures are not particularly effective. Furthermore, while structural connectivity usually implies content-based homophily, the reverse is not always true. Therefore, the use of content similarity has mixed results in different network domains. On the other hand, structural measures are almost always effective in different types of networks. This is because triadic closure is ubiquitous across different network domains and has more direct applicability to link prediction.

19.5.1 Neighborhood-Based Measures

Neighborhood-based measures use the number of common neighbors between a pair of nodes i and j, in different ways, to quantify the likelihood of a link between them in the future. For example, in Fig. 19.12a, Alice and Bob share four common neighbors. Therefore, it is reasonable to conjecture that a link might eventually form between them. In addition to their common neighbors, they also have their own disjoint sets of neighbors. There are different ways of normalizing neighborhood-based measures to account for the number and relative importance of different neighbors. These are discussed below.

Definition 19.5.1 (Common Neighbor Measure) *The common-neighbor measure between nodes i and j is equal to the number of common neighbors between nodes i and j. In other words, if S_i is the neighbor set of node i, and S_j is the neighbor set of node j, the common-neighbor measure is defined as follows:*

$$CommonNeighbors(i, j) = |S_i \cap S_j| \tag{19.47}$$

The major weakness of the common-neighbor measure is that it does not account for the *relative* number of common neighbors between them as compared to the number of other connections. In the example of Fig. 19.12a, Alice and Bob each have a relatively small node degree. Consider a different case in which Alice and Bob are either spammers or very popular public figures who were connected to a large number of other actors. In such a case, Alice and Bob might easily have many neighbors in common, *just by chance*. The *Jaccard measure* is designed to normalize for varying degree distributions.

Definition 19.5.2 (Jaccard Measure) *The Jaccard-based link prediction measure between nodes i and j is equal to the Jaccard coefficient between their neighbor sets S_i and S_j, respectively.*

$$JaccardPredict(i, j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} \tag{19.48}$$

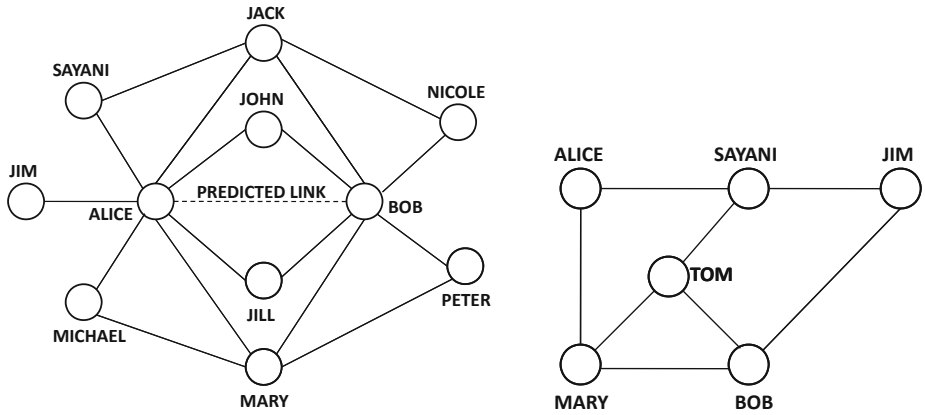
The Jaccard measure between Alice and Bob in Fig. 19.12(a) is 4/9. If the degrees of either Alice or Bob were to increase, it would result in a lower Jaccard coefficient between them. This kind of normalization is important, because of the power-law degree distributions of nodes.

The Jaccard measure adjusts much better to the variations in the degrees of the nodes *between which* the link prediction is measured. However, it does not adjust well to the degrees of their *intermediate* neighbors. For example, in Fig. 19.12a, the common neighbors of Alice and Bob are Jack, John, Jill, and Mary. However, all these common neighbors could be very popular public figures with very high degrees. Therefore, these nodes are statistically more likely to occur as common neighbors of many pairs of nodes. This makes them *less* important in the link prediction measure. The Adamic–Adar measure is designed to account for the varying importance of the different common neighbors. It can be viewed as a weighted version of the common-neighbor measure, where the weight of a common neighbor is a decreasing function of its node degree. The typical function used in the case of the Adamic–Adar measure is the inverse logarithm. In this case, the weight of the common neighbor with index k is set to $1/\log(|S_k|)$, where S_k is the neighbor set of node k.

Definition 19.5.3 (Adamic–Adar Measure) *The common-neighbor measure between nodes i and j is equal to the weighted number of common neighbors between nodes i and j. The weight of node k is defined is $1/log(|S_k|)$.*

$$AdamicAdar(i, j) = \sum_{k \in S_i \cap S_j} \frac{1}{log(|S_k|)} \tag{19.49}$$

The base of the logarithm does not matter in the previous definition, as long as it is chosen consistently for all pairs of nodes. In Fig. 19.12a, the Adamic-Adar measure between Alice and Bob is $\frac{1}{\log(4)} + \frac{1}{\log(2)} + \frac{1}{\log(2)} + \frac{1}{\log(4)} = \frac{3}{\log(2)}$.



(a) Many common neighbors between Alice and Bob

(b) Many indirect connections between Alice and Bob

Figure 19.12: Examples of varying effectiveness of different link-prediction measures

19.5.2 Katz Measure

While the neighborhood-based measures provide a robust estimation of the likelihood of a link forming between a pair of nodes, they are not quite as effective when the number of shared neighbors between a pair of nodes is small. For example, in the case of Fig. 19.12b, Alice and Bob share one neighbor in common. Alice and Jim also share one neighbor in common. Therefore, neighborhood-based measures have difficulty in distinguishing between different pairwise prediction strengths in these cases. Nevertheless, there also seems to be a significant *indirect* connectivity in these cases through longer paths. In such cases, walk-based measures are more appropriate. A particular walk-based measure that is used commonly to measure the link-prediction strength is the *Katz* measure.

Definition 19.5.4 (Katz Measure) *Let $n_{ij}^{(t)}$ be the number of walks of length t between nodes i and j. Then, for a user-defined parameter $\beta < 1$, the Katz measure between nodes i and j is defined as follows:*

$$Katz(i, j) = \sum_{t=1}^{\infty} \beta^t \cdot n_{ij}^{(t)} \tag{19.50}$$

The value of β is a discount factor that de-emphasizes walks of longer length. For small enough values of β, the infinite summation of Eq. 19.50 will converge. If A is the symmetric adjacency matrix of an undirected network, then the $n \times n$ pairwise Katz coefficient matrix K can be computed as follows:

$$K = \sum_{i=1}^{\infty} (\beta A)^i = (I - \beta A)^{-1} - I \tag{19.51}$$

The eigenvalues of A^k are the kth powers of the eigenvalues of A (cf. Eq. 19.33). The value of β should always be selected to be smaller than the inverse of the largest eigenvalue of A to ensure convergence of the infinite summation. A weighted version of the measure can

be computed by replacing A with the weight matrix of the graph. The Katz measure often provides prediction results of excellent quality.

It is noteworthy that the sum of the Katz coefficients of a node i with respect to other nodes is referred to as its *Katz centrality*. Other mechanisms for measuring centrality, such as closeness and *PageRank*, are also used for link prediction in a modified form. The reason for this connection between centrality and link-prediction measures is that highly central nodes have the propensity to form links with many nodes.

19.5.3 Random Walk-Based Measures

Random walk-based measures are a different way of defining connectivity between pairs of nodes. Two such measures are *PageRank* and *SimRank*. Because these methods are described in detail in Sect. 18.4.1.2 of Chap. 18, they will not be discussed in detail here.

The first way of computing the similarity between nodes i and j is with the use of the personalized *PageRank* of node j, where the restart is performed at node i. The idea is that if j is the structural proximity of i, it will have a very high personalized *PageRank* measure, when the restart is performed at node i. This is indicative of higher link prediction strength between nodes i and j. The personalized *PageRank* is an asymmetric measure between nodes i and j. Because the discussion in this section is for the case of undirected graphs, one can use the average of the values of $PersonalizedPageRank(i, j)$ and $PersonalizedPageRank(j, i)$. Another possibility is the *SimRank* measure that is already a symmetric measure. This measure computes an inverse function of the walk length required by two random surfers moving backwards to meet at the same point. The corresponding value is reported as the link prediction measure. Readers are advised to refer to Sect. 18.4.1.2 of Chap. 18 for details of the *SimRank* computation.

19.5.4 Link Prediction as a Classification Problem

The aforementioned measures are *unsupervised* heuristics. For a given network, one of these measures might be more effective, whereas another might be more effective for a different network. How can one resolve this dilemma and select the measures that are most effective for a given network?

The link prediction problem can be viewed as a classification problem by treating the presence or absence of a link between a pair of nodes as a binary class indicator. Thus, a multidimensional data record can be extracted for *each pair of nodes*. The features of this multidimensional record include all the different neighborhood-based, Katz-based, or walk-based similarities between nodes. In addition, a number of other preferential-attachment features, such as node-degrees of each node in the pair, are used. Thus, for each node pair, a multidimensional data record is constructed. The result is a positive-unlabeled classification problem, where node pairs with edges are the positive examples, and the remaining pairs are unlabeled examples. The unlabeled examples can be approximately treated as negative examples for training purposes. Because there are too many negative example pairs in large and sparse networks, only a sample of the negative examples is used. Therefore, the supervised link prediction algorithm works as follows:

1. *Training phase:* Generate a multidimensional data set containing one data record for each pair of nodes with an edge between them, and a sample of data records from pairs of nodes without edges between them. The features correspond to extracted similarity and structural features between node pairs. The class label is the presence or absence of an edge between the pair. Construct a training model on the data.

2. *Testing phase:* Convert each test node pair to a multidimensional record. Use any conventional multidimensional classifier to make label predictions.

The logistic regression method of Sect. 10.6 in Chap. 10 is a common choice for the base classifier. Cost-sensitive versions of various classifiers are commonly used because of the imbalanced nature of the underlying classification problem.

One advantage of this approach is that content features can be used in a seamless way. For example, the content similarity between a pair of nodes can be used. The classifier will automatically learn the relevance of these features in the training process. Furthermore, unlike many link prediction methods, the approach can also handle *directed networks* by extracting features in an asymmetric way. For example, instead of using node degrees, one might use indegrees and outdegrees as features. Random walk features can also be defined in an asymmetric way on directed networks, such as computing the *PageRank* of node j with restart at node i, and vice versa. In general, the supervised model is more flexible because of its ability to *learn* relationships between links and features of various types.

19.5.5 Link Prediction as a Missing-Value Estimation Problem

Section 18.5.3 of Chap. 18 discusses how link prediction can be applied to user-item graphs for recommendations. In general, both the recommendation problem and the link prediction problem may be viewed as instances of missing value estimation on matrices of different types. Recommendation algorithms are applied to user-item utility matrices, whereas link prediction algorithms are applied to incomplete adjacency matrices. All the 1s in the matrix correspond to edges. Only a small random sample of the remaining entries are set to 0, and the other entries are assumed to be unspecified. Any of the missing-value estimation methods discussed in Sect. 18.5 of Chap. 18 may be used to estimate the values of the missing entries. Among this class of methods, matrix factorization methods are among the most commonly used methods. One advantage of using these methods is that the specified matrix does not need to be symmetric. In other words, the approach can also be used for directed graphs. Refer to the bibliographic notes.

19.5.6 Discussion

The different measures have been shown to have varying levels of effectiveness over different data sets. The advantage of neighborhood-based measures is that they can be computed efficiently for very large data sets. Furthermore, they perform almost as well as the other unsupervised measures. Nevertheless, random walk-based and Katz-based measures are particularly useful for very sparse networks, in which the number of common neighbors cannot be robustly measured. Although supervision provides better accuracy, it is computationally expensive. However, supervision provides the greatest adaptability across various domains of social networks, and available side information such as content features.

In recent years, content has also been used to enhance link prediction. While content can significantly improve link prediction, it is important to point out that structural measures are far more powerful. This is because structural measures directly use the triadic properties of real networks. The triadic property of networks is true across virtually all data domains. On the other hand, content-based measures are based on "reverse homophily," where similar or link-correlated content is leveraged for predicting links. The effectiveness of this is highly network domain-specific. Therefore, content-based measures are often used in a helping role for link prediction and are rarely used in isolation for the prediction process.

19.6 Social Influence Analysis

All social interactions result in varying levels of influence between individuals. In traditional social interactions, this is sometimes referred to as "word of mouth" influence. This general principle is also true for online social networks. For example, when an actor tweets a message in Twitter, the followers of the actors are exposed to the message. The followers may often retweet the message in the network. This results in the spread of information, ideas, and opinions in the social network. Many companies view this kind of information spread as a valuable advertising channel. By tweeting a popular message to the right participants, millions of dollars worth of advertising can be generated, if the message spreads through the social network as a *cascade*. An example [532] is the famous Oreo Superbowl tweet on February 3, 2013. The power went out during the Superbowl game between the San Francisco 49ers and the Baltimore Ravens. Oreo used this opportunity to tweet the following message, along with a picture of an Oreo cookie, during the 34 min interruption: "Power out? No problem. You can still dunk in the dark." Viewers loved Oreo's message, and retweeted it thousands of times. Oreo was thus able to generate millions of dollars of advertising at zero cost, and apparently had a higher impact than *paid* television advertisements during the Superbowl.

Different actors have different abilities to influence their peers in the social network. The two most common factors that regulate the influence of an actor are as follows:

1. Their centrality within the social network structure is a crucial factor in their influence level. For example, actors with high levels of centrality are more likely to be influential. In directed networks, actors with high prestige are more likely to be influential. These measures are discussed in Sect. 19.2.

2. The edges in the network are often associated with weights that are dependent on the likelihood that the corresponding pair of actors can be influenced by each other. Depending on the diffusion model used, these weights can sometimes be directly interpreted as *influence propagation probabilities*. Several factors may determine these probabilities. For example, a well-known individual may have higher influence than lesser known individuals. Similarly, two individuals, who have been friends for a long time, are more likely to influence one another. It is often assumed that the influence propagation probabilities are already available for analytical purposes, although a few recent methods show how to estimate these probabilities in a data-driven way.

The precise impact of the aforementioned factors is quantified with the use of an influence propagation model. These are also referred to as *diffusion models*. The main goal of such models is to determine a set of seed nodes in the network, at which the dissemination of information maximizes influence. Therefore, the influence maximization problem is as follows:

Definition 19.6.1 (Influence Maximization) *Given a social network $G = (N, A)$, determine a set of k seed nodes S, influencing which will maximize the overall spread of influence in the network.*

The value of k can be viewed as a budget on the number of seed nodes that one is allowed to initially influence. This is quite consistent with real-life models, where advertisers are faced with budgets on initial advertising capacity. The goal of social influence analysis is to extend this initial advertising capacity with word-of-mouth methods.

Each model or heuristic can quantify the influence level of a node with the use of a function of S that is denoted by $f(\cdot)$. This function maps subsets of nodes to real numbers representing influence values. Therefore, after a model has been chosen for quantifying the influence $f(S)$ of a *given* set S, the optimization problem is that of determining the set S that maximizes $f(S)$. An interesting property of a very large number of influence analysis models is that the optimized function $f(S)$ is *submodular*.

What does submodularity mean? It is a mathematical way of representing the natural law of diminishing returns, as applied to sets. In other words, if $S \subseteq T$, then the *additional* influence obtained by adding an individual to set T cannot be larger than the additional influence of adding the same individual to set S. Thus, the *incremental* influence of the same individual diminishes, as larger supersets of cohorts are available as seeds. The submodularity of set S is formally defined as follows:

Definition 19.6.2 (Submodularity) *A function $f(\cdot)$ is said to be submodular, if for any pair of sets S, T satisfying $S \subseteq T$, and any set element e, the following is true:*

$$f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T) \tag{19.52}$$

Virtually all natural models for quantifying influence turn out to be submodular. Submodularity is algorithmically convenient because a very efficient greedy optimization algorithm exists for maximizing submodular functions, as long as $f(S)$ can be evaluated for a given value of S. This algorithm starts by setting $S = \{\}$ and incrementally adds nodes to S that increase the value of $f(S)$ as much as possible. This procedure is repeated until the set S contains the required number of influencers k. The approximation level of this heuristic is based on a well-known classical result on optimization of submodular functions.

Lemma 19.6.1 *The greedy algorithm for maximizing submodular functions provides a solution with an objective function value that is at least a fraction $\left(\frac{e-1}{e}\right)$ of the optimal value. Here, e is the base of the natural logarithm.*

Thus, these results show that it is possible to optimize $f(S)$ effectively, as long as an appropriate submodular influence function $f(S)$ can be defined for a given set of nodes S.

Two common approaches for defining the influence function $f(S)$ of a set of nodes S are the *Linear Threshold Model* and the *Independent Cascade Model*. Both these diffusion models were proposed in one of the earliest works on social influence analysis. The general operational assumption in these diffusion models is that nodes are either in an active or inactive state. Intuitively, an *active* node is one which has already been influenced by the set of desired behaviors. Once a node moves to an active state, it never deactivates. Depending on the model, an active node may trigger activation of neighboring nodes either for a single time, or over longer periods. Nodes are successively activated until no more nodes are activated in a given iteration. The value of $f(S)$ is evaluated as the total number of activated nodes at termination.

19.6.1 Linear Threshold Model

In this model, the algorithm initially starts with an active set of seed nodes S and iteratively increases the number of active nodes based on the influence of neighboring active nodes. Active nodes are allowed to influence their neighbors over multiple iterations throughout the execution of the algorithm until no more nodes can be activated. The influence of

neighboring nodes is quantified with the use of a linear function of the edge-specific weights b_{ij}. For each node i in the network $G = (N, A)$, the following is assumed to be true:

$$\sum_{j:(i,j)\in A} b_{ij} \leq 1 \tag{19.53}$$

Each node i is associated with a random threshold $\theta_i \sim U[0, 1]$ which is fixed up front and stays constant over the course of the algorithm. The total influence $I(i)$ of the *active* neighbors of node i on it, at a given time-instant, is computed as the sum of the weights b_{ij} of all *active* neighbors of i.

$$I(i) = \sum_{j:(i,j)\in A, j \text{ is active}} b_{ij} \tag{19.54}$$

The node i becomes active in a step when $I(i) \geq \theta_i$. This process is repeated until no further nodes can be activated. The total influence $f(S)$ may be measured as the number of nodes activated by a given seed set S. The influence $f(S)$ of a given seed set S is typically computed with simulation methods.

19.6.2 Independent Cascade Model

In the aforementioned linear threshold model, once a node becomes active, it has multiple chances to influence its neighbors. The random variable θ_i was associated with a *node*, in the form of a threshold. On the other hand, in the independent cascade model, after a node becomes active, it obtains only a *single chance* to activate its neighbors, with *propagation probabilities* associated with the *edges*. The propagation probability associated with an edge is denoted by p_{ij}. In each iteration, only the *newly* active nodes are allowed to influence their neighbors, that have not already been activated. For a given node j, each of the edges (i, j) joining it to its newly active neighbors i flips a coin independently with success probability p_{ij}. If the coin toss for edge (i, j) results in a success, then the node j is activated. If node j is activated, it will get a single chance in the next iteration to influence its neighbors. In the event that no nodes are newly activated in an iteration, the algorithm terminates. The influence function value is equal to the number of active nodes at termination. Because nodes are allowed to influence their neighbors only once over the course of the algorithm, a coin is tossed for each edge at most once over the course of the algorithm.

19.6.3 Influence Function Evaluation

Both the linear threshold model and the independent cascade model are designed to compute the influence function $f(S)$ with the use of a model. The estimation of $f(S)$ is typically accomplished with simulation.

For example, consider the case of the linear threshold model. For a given seed node set S, one can use a random number generator to set the thresholds at the nodes. After the thresholds have been set, the active nodes can be labeled using any deterministic graph-search algorithm starting from the seed nodes in S and progressively activating nodes when the threshold condition is satisfied. The computation can be repeated over different sets of randomly generated thresholds, and the results may be averaged to obtain more robust estimates.

In the independent cascade model, a different simulation may be used. A coin with probability p_{ij} may be flipped for each edge. The edge is designated as *live* if the coin toss was a success. It can be shown that a node will eventually be activated by the independent

cascade model, when a path of live edges exists from at least one node in S to it. This can be used to estimate the size of the (final) active set by simulation. The computation is repeated over different runs and the results are averaged.

The proof that the linear threshold model and the independent cascade model are submodular optimization problems can be found in pointers included in the bibliographic notes. However, this property is not specific to these models. Submodularity is a very natural consequence of the laws of diminishing returns, as applied to the incremental impact of individual influence in larger groups. As a result, most reasonable models for influence analysis will satisfy submodularity.

19.7 Summary

Social networks have become increasingly popular in recent years, because of their ability to connect geographically and culturally diverse participants. A significant amount of data is created because of the actions of social network participants. Much of this data are structural, in the form of relationships between different individuals.

Social network structures exhibit a number of typical properties, because of the natural dynamics of their formation. The most important similarity-based properties include triadic closure, and homophily. Typically, social networks are formed by preferential attachment, and they exhibit power-law degree distributions.

The problem of clustering social networks is challenging because of the presence of hub nodes, and the natural tendency of social networks to cluster into a single large group. Therefore, most community detection algorithms have built-in mechanisms to ensure that the underlying clusters are balanced. Clustering methods are also sometimes referred to as graph-partitioning. One of the earliest clustering methods was the Kernighan–Lin method, which uses an iterative approach for clustering. Nodes are repeatedly exchanged between partitions to iteratively improve the value of the objective function. The Girvan–Newman algorithm uses notions of betweenness centrality to generate clusters. The *METIS* algorithm generates an efficient partition by using coarsening and then creating the partitions on the coarsened representation. The spectral method uses multidimensional embeddings to generate the clusters.

In collective classification, the goal is to infer labels at the remaining vertices from the pre-existing labels at a subset of the vertices. This is a problem that has dual applicability to social network analysis and semisupervised learning. Multidimensional data sets can be transformed into similarity graphs to apply collective classification methods. The most common methods used for collective classification include iterative methods, random walk-based label propagation methods, and spectral methods.

In the link-prediction problem, the goal is to predict the links from the currently available structure and content in the network. Structural measures are generally much more effective for link-prediction than content-based measures. The structural methods use local clustering measures such as the Jaccard measure or personalized *PageRank* values for making predictions. Supervised methods are able to discriminatively determine the most relevant features for link prediction.

Social networks are often used for influencing individuals using "word-of-mouth" techniques. Typically, centrally located actors are more influential in the network. Diffusion models are used to characterize the flow of information in social networks. Two examples of such models include the linear threshold model and the independent cascade model.

19.8 Bibliographic Notes

Social network analysis has been studied extensively in the context of the field of sociology [508], though more recent work has focused on online social networks [6, 192, 532]. A detailed discussion on proximity and centrality measures may be found in [6, 192, 508, 532]. The dynamics of social network formation may be found in the excellent survey paper [69]. The derivation of the power-law with the use of the scale-free model is provided in [70]. A detailed study of the power-law in the context of the Internet topology is provided in [201]. A study of graph densification and shrinking diameters is provided in [342]. Other random graph models such as the Erdos–Renyi model and the Watts–Strogatz small-world model are discussed in [196, 509].

A detailed survey on community detection methods may be found in [212]. The minimum cut problem is polynomially solvable for some special cases. For example, the unweighted 2-way cut problem is polynomially solvable without balancing constraints [299]. The original Kernighan–Lin algorithm is presented in [312]. The enhancements to the Kernighan–Lin algorithm was discussed in [206, 301]. The Girvan–Newman algorithm discussed in this chapter is adapted from [230]. The *METIS* algorithm is presented in [301]. The normalized cut method for spectral clustering was discussed in this chapter [466]. The normalized symmetric version was proposed in [405]. More details on spectral graph theory and clustering methods may be found in [152, 371]. This chapter uses the *Laplacian eigenmap* interpretation [90] of spectral clustering, rather than the more commonly used cut interpretation, because of its comprehensive explanation of the non-integer and possibly negative eigenvector components.

ICA has been presented in the context of many different data domains, such as document data [128], and relational data [404]. Several base classifiers have been used within this framework, such as logistic regression [370] and a weighted voting classifier [373]. The discussion in this chapter is based on [404]. The iterative label propagation method was proposed in [554], and the absorbing random walk interpretation is adapted from [78]. The iterative label propagation approach [554] was originally proposed with a spectral interpretation although the random walk interpretation is also briefly discussed in the same work. Most random walk methods can also be formulated as supervised versions of spectral embeddings [530, 551, 554]. The regularization framework for collective classification is discussed in [551]. Collective classification of directed graphs is discussed in [552]. A method for incorporating content within the random walk framework is discussed in [44]. Detailed surveys on node classification methods may be found in [93, 368]. A toolkit for collective classification may be found in [427].

The link-prediction problem for social networks was proposed in [353]. The measures discussed in this chapter are based on this work. Since then, a significant amount of work has been done on incorporating content into the link prediction process. Methods that use content for link prediction may be found in [49, 64, 354, 484, 489]. The merits of supervised methods are discussed in [354], and matrix factorization methods are discussed in [383]. Recently, it has been shown how to use link prediction across multiple networks in [428]. A survey on link-prediction methods for social network analysis may be found in [63].

The problem of influence analysis in social networks was proposed in [304]. The linear threshold and independent cascade models are presented in this work. The degree-discount heuristic was proposed in [142]. A discussion of the submodularity property may be found in [403]. Other recent models for influence analysis in social networks are discussed in [45, 143, 144, 362, 488]. One of the main problems in social influence models is a difficulty in learning the influence propagation probabilities, though there has been some recent focus

on this issue [235]. Recent work has also shown how influence analysis can be performed directly from the social stream [234, 482]. A survey on models and algorithms for social influence analysis may be found in [483].

19.9 Exercises

1. For the figure in Example 19.1a, compute the highest-degree centrality, closeness centrality and betweenness centrality. The nodes that take on these highest values are already marked in the figure.

2. Implement the algorithms for determining the degree centrality, closeness centrality, and betweenness centrality.

3. Implement the Kernighan–Lin algorithm.

4. Why is the balancing constraint more important in community detection algorithms, as compared to multidimensional clustering algorithms? What would the unconstrained minimum 2-way cut look like in a typical real network?

5. Consider a variation of Girvan–Newman algorithm in which edges are randomly disconnected from a network, as opposed to those with high betweenness centrality. Explain the negative impact of this change on the algorithm. Can you make minor changes to the disconnection criterion to ameliorate this impact?

6. Write an integer-programming formulation for the minimum 2-way cut problem, so that the cut is balanced in terms of the number of nodes.

7. For the random walk formulation of spectral clustering algorithm, show why the following are true:

   (a) All nontrivial eigenvectors $\overline{y}$ have both positive and negative components.

   (b) Provide an intuitive explanation why the normalization factor Λ in the constraint $\overline{y}^T \Lambda \overline{y} = 1$, increases the propensity of low-degree nodes to be embedded away from the origin, and the high-degree nodes to be embedded near the origin.

8. Suppose that all edge weights w_{ij} are discounted by the geometric mean of the weighted node-degrees at their endpoints. Write an unnormalized formulation of spectral clustering in terms of these normalized weights for discovering a 1-dimensional embedding. What effect would the weight normalization have on the embedding? Describe the algebraic similarities and differences of this formulation from the symmetric normalized formulation of spectral clustering. Discuss why the resulting eigenvectors will often be heuristically similar to those obtained with the symmetric formulation of spectral clustering.

9. Explain the relationship between the random walk label propagation and the graph regularization algorithm.

10. Discuss the connections between the link-prediction problem and network clustering.

11. Create a link prediction measure that can perform the degree normalizations performed both by the Jaccard measure and the Adamic–Adar measure.

12. Implement the linear threshold and independent cascade model for influence analysis.

13. The chapter provides a 1-dimensional formulation for the symmetric version using the column vector $\overline{z}$. Set up a generalized formulation for the symmetric version using an $n \times k$ matrix Z.

(a) Let Y be the decision variables for the random walk formulation discussed in the chapter. Show that $Z = \sqrt{\Lambda}Y$.

(b) Show that the unit-norm scaled rows of Y and Z are the same.

14. It is well known that a symmetric matrix always has real eigenvalues. Use this result to show that the stochastic transition matrix of an *undirected* graph always has real eigenvalues.

15. Show that if $(\overline{y}, \lambda)$ is an eigenvector–eigenvalue pair of the normalized Laplacian $\Lambda^{-1}(\Lambda - W)$, then $(\overline{y}, 1-\lambda)$ is an eigenvector–eigenvalue pair of the normalized weight matrix $\Lambda^{-1}W$. Here, Λ is a diagonal matrix containing the sum of each row in the weighted adjacency matrix W.