Community Detection

Simone Santoni

2024-11-26

Synopsis This notebook shows communities in a network — that is, groups of nodes densely connected to each others and sparsely connected with outgroup nodes. Specifically, the attention revolves around two popular community detection algorithms like Girvan-Newman and Louvain's.

1 Notebook setup

For this tutorial, we rely on 'usual suspects' Python packages, like `numpy`, `matplotlib`, and `networkx`. The latter is the most popular Python package for the creation, manipulation, and study of the structure small to moderate size networks.

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
```

2 Load Karate Club network

The Karate Club dataset is a well-known social network dataset representing the friendships between 34 members of a karate club at a US university in the 1970s[1]. The network consists of 34 nodes and 78 edges, where nodes represent members and edges represent friendships. The dataset is often used for testing community detection algorithms, as it naturally splits into two communities due to a conflict between the club's instructor and the administrator, leading to the formation of two separate clubs.

```
G = nx.karate_club_graph()
```

[1] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. Journal of anthropological research, 33(4), 452-473. doi:10.1086/jar.33.4.3629752

3 Visualize the network

The visual inspection of the network (see Figure 1) reveals two distinct groups of nodes that may correspond to two communities, i.e., groups of nodes that are more densely connected to each other than to nodes outside the group. Communities often represent functional units within the network, such as groups of friends in a social network, modules in a biological network, or clusters of related documents in an information network. However, we need to produce conclusive evidence that these groups are indeed communities.

```
# fix node positions for better visualization
pos = nx.spring_layout(G, seed=123)
# draw the network
nx.draw(
         G,   pos,   with_labels=True,   node_color="lightgray",   node_size=300,
edge_color="gray"
)
```

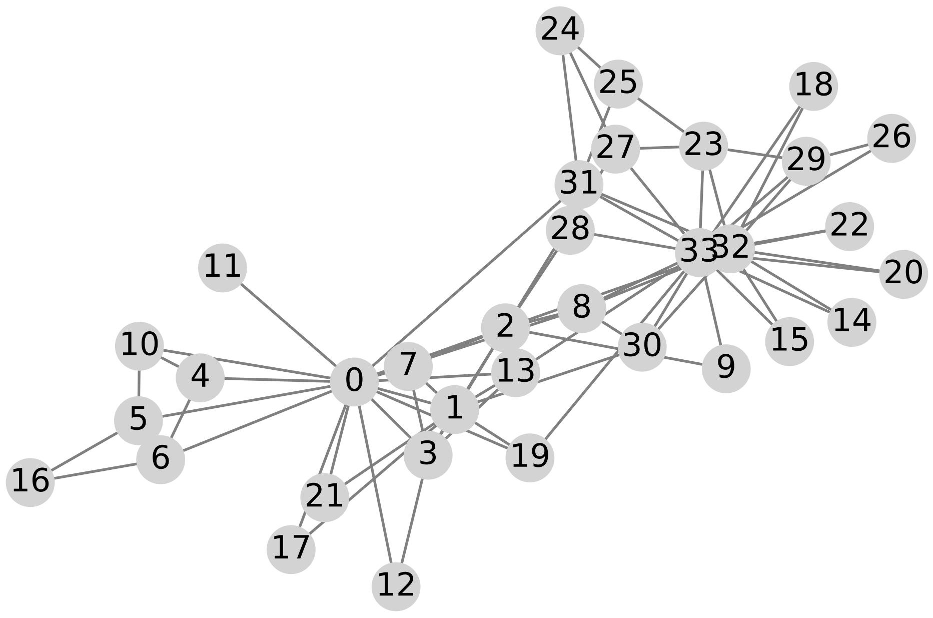

Figure 1: Visualization of the Karate Club network

4 Community detection using Girvan-Newman's algorithm

`networkx` provides an implementation of the Girvan-Newman[2] algorithm, which is a hierarchical clustering method based on edge betweenness centrality. The algorithm iteratively removes the edge with the highest betweenness centrality, recalculates the centrality of the remaining edges,

[2]Girvan, M., & Newman, M. E. J. (2002). Community structure in social and biological networks. Proceedings of the National Academy of Sciences, 99(12), 7821-7826. doi:10.1073/pnas.122653799

and identifies the connected components of the graph. The process continues until the desired number of communities is reached.

Let us consider the first iteration of the Girvan-Newman algorithm, which consists of computing edge betweenness centrality. In Figure 2), the edges are color-coded against their betweenness centrality values, with warmer colors indicating higher centrality.

```
# edge betweenness centrality
edge_betweenness = nx.edge_betweenness_centrality(G)
# network visualization
nx.draw(
    G,
    pos,
    with_labels=True,
    node_color="lightgray",
    node_size=300,
    edgelist=edge_betweenness.keys(),
    edge_color=list(edge_betweenness.values()),
    edge_cmap=plt.cm.Reds,
    edge_vmin=0,
    edge_vmax=0.1,
)
```

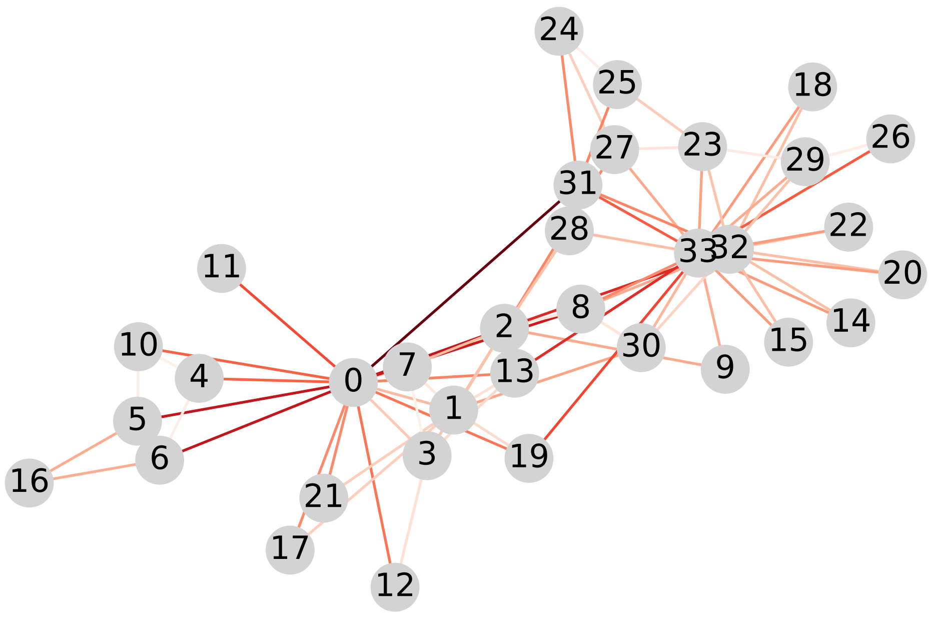

Figure 2: Edge betweenness centrality in the Karate Club network

The visual inspection of edge betweenness centrality suggests that the edge connecting nodes `0` and `31` has the highest centrality. We can check this by sorting the edges by centrality and examining the top five edges.

```
edge_betweenness_sorted = sorted(edge_betweenness.items(), key=lambda x: x[1],
reverse=True)
print(edge_betweenness_sorted[:5])
```

```
[((0, 31), 0.12725999490070537), ((0, 6), 0.07813428401663695), ((0,
5), 0.07813428401663694), ((0, 2), 0.0777876807288572), ((0, 8),
0.07423959482783014)]
```

The second step consists of removing the `0-31` and recalculating the centrality of the remaining edges. It is straight-forward that `G` will still be connected. In other words, we will not be able to see the two groups of nodes that get disconnected because of the removal one specific edge. Therefore, we will not have identified any partitioning of the network, that is, community structure. The process is repeated until the network breaks down into two connected components least.

```
# remove edge 0-31
G.remove_edge(0, 31)
# recalculate edge betweenness centrality
edge_betweenness = nx.edge_betweenness_centrality(G)
# inspect the first 5 edges by centrality
edge_betweenness_sorted = sorted(edge_betweenness.items(), key=lambda x: x[1],
reverse=True)
print(edge_betweenness_sorted[:5])
# double check that the graph is still connected
print(nx.is_connected(G))
```

```
[((0, 2), 0.11924273983097515), ((0, 8), 0.09923105217222859), ((2,
32), 0.08791752909399968), ((13, 33), 0.08660576895871015), ((0, 5),
0.07813428401663694)]
True
```

Figure 3 visualizes the network after removing the edge `0-31`. The two communities are clearly visible, with nodes `0` and `31` belonging to different groups.

```
nx.draw(
    G,
    pos,
    with_labels=True,
    node_color="lightgray",
    node_size=300,
    edgelist=edge_betweenness.keys(),
    edge_color=list(edge_betweenness.values()),
    edge_cmap=plt.cm.Reds,
    edge_vmin=0,
    edge_vmax=0.1,
)
```

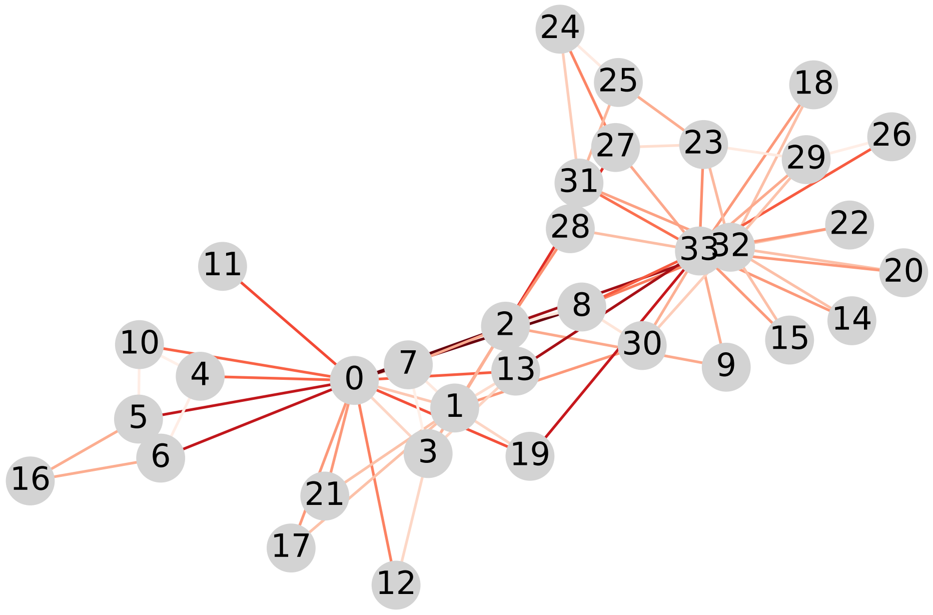

Figure 3: Visualization of the Karate Club network after removing the edge 0-31

The intuition behind the Girvan-Newman algorithm is that edges connecting different communities have higher betweenness centrality, as they are crucial for connecting the communities. By iteratively removing these edges, the algorithm effectively identifies the communities in the network. For example, Figure 3 shows the `G` is at risk to get disconnected if edges like `0-2`, `0-8`, and `19-33` are removed.

Luckily, `networkx` provides a convenient function `community.girvan_newman` to automate the process of community detection using the Girvan-Newman algorithm. The function returns an iterator over the discovered communities, allowing us to stop the algorithm at a specific number of communities. Let us apply the Girvan-Newman algorithm to the Karate Club network and visualize the communities.

```
# we must re-add the edge 0-31 to the graph
G.add_edge(0, 31)
# Girvan-Newman algorithm
fit = nx.community.girvan_newman(G)
tuple(sorted(c) for c in next(fit))
```

```
([0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 19, 21],
 [2, 8, 9, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33])
```

By default, the Girvan-Newman algorithm stops when the graph is partitioned into two communities. However, we can specify the desired number of communities by stopping the algorithm at a specific level. For example, we can stop the algorithm at the third level to obtain three communities.

```python
import itertools
k = 4
fit = nx.community.girvan_newman(G)
limited = itertools.takewhile(lambda c: len(c) <= k, fit)
for communities in limited:
    print(tuple(sorted(c) for c in communities))
```

```
([0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 19, 21], [2, 8, 9, 14, 15, 18,
20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33])
([0, 1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 19, 21], [2, 8, 14, 15, 18, 20,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33], [9])
([0, 1, 3, 7, 11, 12, 13, 17, 19, 21], [2, 8, 14, 15, 18, 20, 22, 23, 24, 25,
26, 27, 28, 29, 30, 31, 32, 33], [4, 5, 6, 10, 16], [9])
```

The visual inspection of Girvan-Newman's algorithm outcome is a plausible place to start to adjudicate between alternative community structures.[3] Let us start by visualizing the network with two communities (see Figure 4).

```python
# fit the Girvan-Newman algorithm
fit = nx.community.girvan_newman(G)
# we retain the first three partitions of the network
k = 4
# get the membership of the nodes into communities
limited = itertools.takewhile(lambda c: len(c) <= k, fit)
fits = {}
for _, communities in enumerate(limited):
    fits[_] = tuple(sorted(c) for c in communities)
# get the membership of the nodes into communities
two_communities = fits[0]
# color code the communities
colors = ["plum" if node in two_communities[0] else "lightgreen" for node in
G.nodes]
# visualize the network
nx.draw(
```

[3]It is worth noticing that Girvan-Newman's algorithm is not deterministic, and the results may vary depending on the initial conditions and the order in which edges are removed. Therefore, it is essential to consider multiple runs of the algorithm and compare the results to identify robust communities.

```
    G,
    pos,
    with_labels=True,
    node_color=colors,
    node_size=300,
    edge_color="gray",
)
```

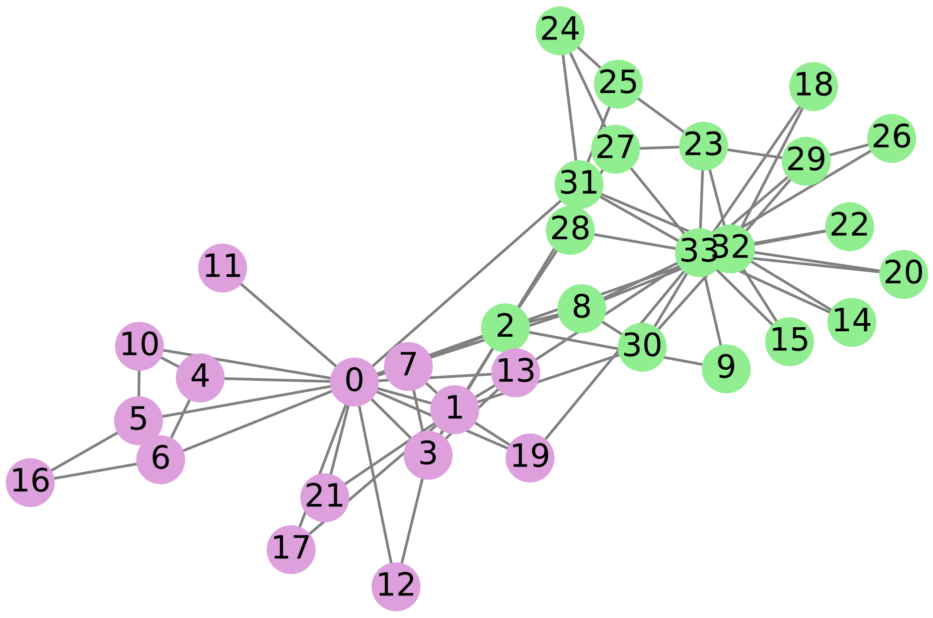

Figure 4: Visualization of the Karate Club network with two communities

One may point out that the solution in Figure 4 presents a clear-cut division of the network into two communities. However, the division is not perfect, as some nodes are on the boundary between the two communities (see for example nodes 2 and 13). This is a common issue in community detection, as nodes can have multiple connections to different communities. The Girvan-Newman algorithm is a divisive method that partitions the network into communities by removing edges, which may lead to suboptimal results.

The presence of boarder nodes is not the most concerning issue in this case, though. The lower-left section of Figure 4 indicates the presence of a group of nodes that are densely connected to each other but are not clearly part of the two main communities. Let us visualize the network with three communities to investigate this further.

```
# color code the communities
three_communities = fits[1]
# print(three_communities)
```

```
colors = [
    (
        "plum" if node in three_communities[0]
        else "lightgreen" if node in three_communities[1]
        else "lightblue"
    )
    for node in G.nodes
]
# visualize the network
nx.draw(
    G,
    pos,
    with_labels=True,
    node_color=colors,
    node_size=300,
    edge_color="gray",
)
```

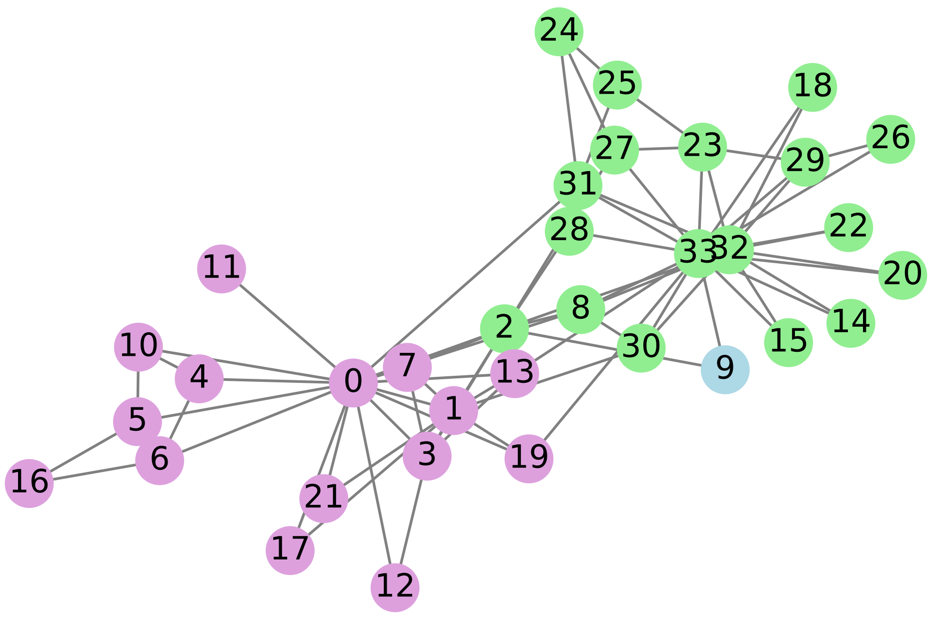

Figure 5: Visualization of the Karate Club network with three communities

The three-community structure does not yield the expected representation of the network (in which nodes `4`, `5`, `6`, `10`, and `16` form their own community). Instead, it is node `9`, a ‘boarder’ node, that gets assigned to the third community. In light of this unsatisfactory solution, one may want to render and visualize the four-community structure (see Figure 6).

```python
# color code the communities
four_communities = fits[2]
# print(three_communities)
colors = [
    (
        "plum" if node in four_communities[0]
        else "lightgreen" if node in four_communities[1]
        else "orange" if node in four_communities[2]
        else "lightblue"
    )
    for node in G.nodes
]
# visualize the network
nx.draw(
    G,
    pos,
    with_labels=True,
    node_color=colors,
    node_size=300,
    edge_color="gray",
)
```

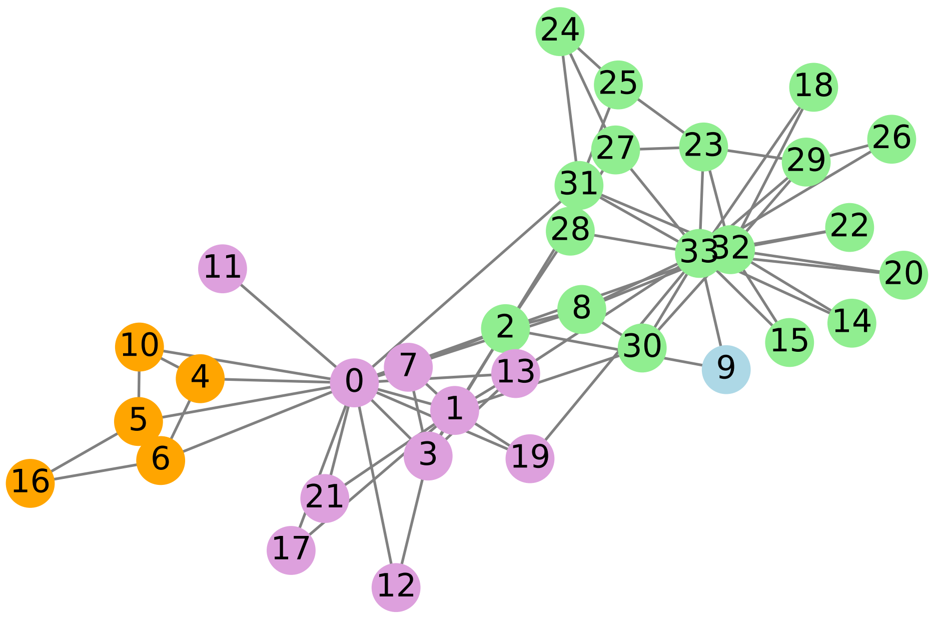

Figure 6: Visualization of the Karate Club network with four communities

5 Community detection using Louvaine's algorithm

The Louvain community detection algorithm[4] is a popular method for identifying communities in large networks. It is an iterative, modularity-based algorithm that optimizes the modularity of a partition of the network[5]. Modularity is a measure of the density of links inside communities compared to links between communities.

The algorithm operates in two main phases that are repeated iteratively. In the first phase, each node is assigned to its own community. Then, for each node, the algorithm considers moving it to the community of each of its neighbors, choosing the move that results in the highest increase (or smallest decrease) in modularity. This process is repeated for all nodes until no further improvement can be achieved.

In the second phase, the algorithm aggregates nodes belonging to the same community into a single node, creating a new, smaller network. Edges between the new nodes are weighted by the sum of the weights of the edges between the original nodes in the corresponding communities. The first phase is then reapplied to this new network.

These two phases are repeated iteratively until the modularity no longer increases significantly. The result is a hierarchical decomposition of the network into communities, which can be represented at different levels of granularity. The Louvain algorithm is efficient and can handle large networks, making it widely used in various applications, including social network analysis, biology, and information retrieval.

Let us consider an example of applying the Louvain algorithm to the Karate Club network. The `community` module in `networkx` provides an implementation of the Louvain algorithm, which we can use to detect communities in the network.

```python
# Louvain algorithm fit
fit = nx.community.louvain_communities(G)
# retriece the communities
communities = tuple(sorted(c) for c in fit)
print(communities)
```

```
([0, 4, 5, 6, 10, 11, 16, 17, 19, 21], [1, 2, 3, 7, 12, 13], [23, 24, 25, 27,
28, 31], [8, 9, 14, 15, 18, 20, 22, 26, 29, 30, 32, 33])
```

The community structure solution that maximizes the modularity criterion comprisese the following communities: `0, 1, 2, 3, 7, 13, 17, 19, 21` and `4, 5, 6, 10, 16` and `8, 9, 11,`

[4]Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10), P10008. doi:10.1088/1742-5468/2008/10/P10008

[5]Nicolas Dugué, Anthony Perez. Directed Louvain : maximizing modularity in directed networks. [Research Report] Université d'Orléans. 2015. hal-01231784. https://hal.archives-ouvertes.fr/hal-01231784

`12, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33`. The following Figure 7 visualize the network with the identified communities.

```
colors = [
    (
        "plum" if node in communities[0]
        else "lightgreen" if node in communities[1]
        else "lightblue"
    )
    for node in G.nodes
]
# visualize the network
nx.draw(
    G,
    pos,
    with_labels=True,
    node_color=colors,
    node_size=300,
    edge_color="gray",
)
```

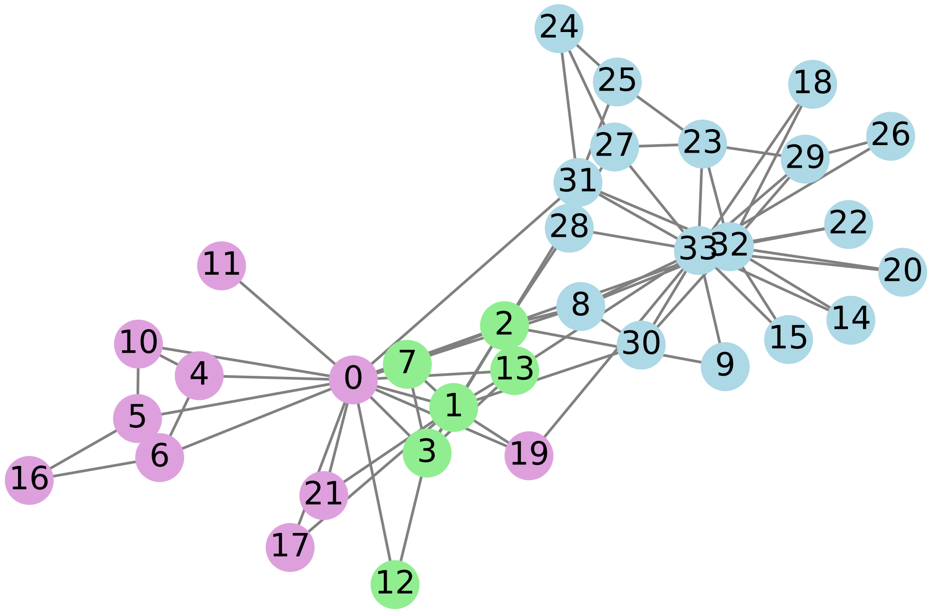

Figure 7: Visualization of the Karate Club network with Louvain communities

The Louvain algorithm is not only capable to isolate the most plausible community structure in a network. It can also handle weighted networks. Let us consider the case of a weighted Karate Club network, where the edge weights represent the strength of the friendship between members (see Figure 8). The following code snippet shows how to create a weighted version of the Karate Club network and apply the Louvain algorithm to detect communities.

```
# weighted Karate Club network
G_weighted = nx.karate_club_graph()
# assign random weights to the edges
import numpy as np
for u, v in G_weighted.edges:
    G_weighted[u][v]["weight"] = np.random.random_integers(1, 10)
# visualize the weighted network
nx.draw(
    G_weighted,
    pos,
    with_labels=True,
    node_color="lightgray",
    node_size=300,
    edge_color=[G_weighted[u][v]["weight"] for u, v in G_weighted.edges],
    edge_cmap=plt.cm.Greens,
    edge_vmin=0,
    edge_vmax=10,
)
```

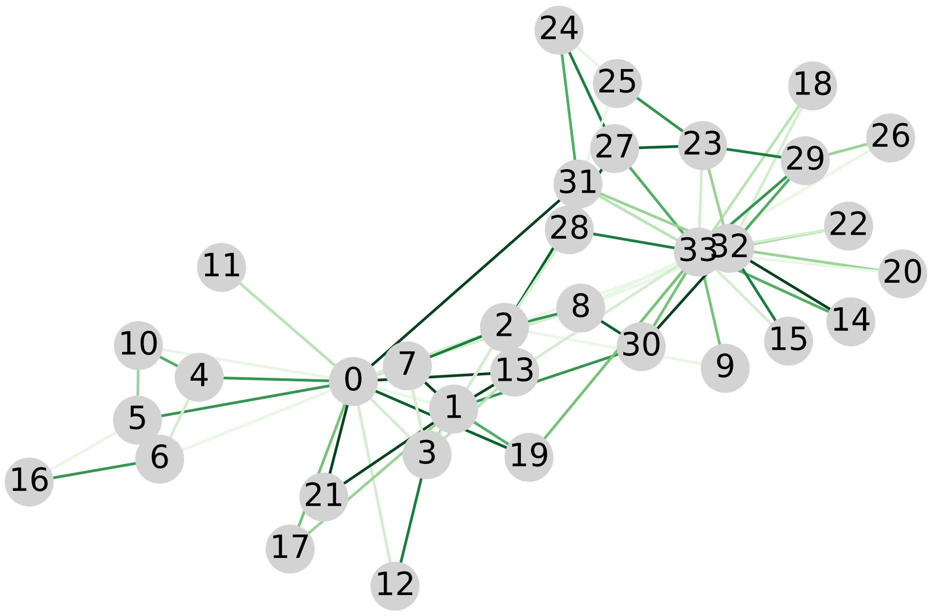

Figure 8: Visualization of the weighted Karate Club network

Then, we fit the Louvain algorithm to the weighted network and visualize the communities — see Figure 9.

```python
# fit the Louvain algorithm to the weighted network
fit = nx.community.louvain_communities(G_weighted, weight="weight")
# retrieve the communities
communities = tuple(sorted(c) for c in fit)
# visualize the network with the identified communities
colors = [
    (
        "plum" if node in communities[0]
        else "lightgreen" if node in communities[1]
        else "lightblue"
    )
    for node in G_weighted.nodes
]
# visualize the network
nx.draw(
    G_weighted,
    pos,
    with_labels=True,
    node_color=colors,
    node_size=300,
    edge_color=[G_weighted[u][v]["weight"] for u, v in G_weighted.edges],
    edge_cmap=plt.cm.Greens,
    edge_vmin=0,
    edge_vmax=10,
)
```

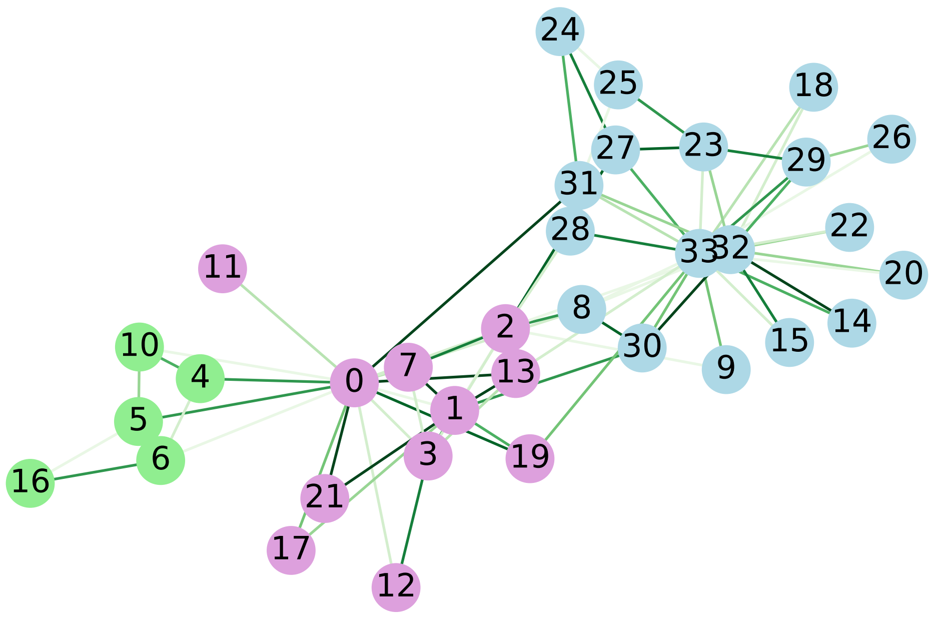

Figure 9: Visualization of the Karate Club network with Louvain communities in the weighted network

Considering the weighted network, the Louvain algorithm yields some notable results:

- The strong ties between nodes `0`, `4`m, and `10` make nodes `0` and `4` part of the same community, despite the redundant ties to nodes `5`, `6` and `15` (compare Figure 7 and Figure 9)
- Nodes located at the boarder of the communities are more likely to be assigned to the community with which they share the strongest ties. For example, node `9` is assigned to the same community as node `30`; node `19` is assigned to to the same community as node `3`.