CREATE YOUR OWN MUSIC WITH COMMUNITY DETECTION ALGORITHM

XINGYU WANG, XIWEN LIU

Abstract. This paper looks into two algorithms, Girvan-Newman and Label Propogation, in finding communities within network structure. With the community detection algorithms, we hope to find patterns in chord progressions of different music and generate our own music. Most related projects and research focus on network structure of music in the same genre, while we utilize songs from different genres as a demonstration in the paper. Additionally, the performance of Girvan-Newman and Label Propagation is compared in this paper.

1. Introduction

Many, even cross genre, music pieces have similar chord progression. Being inspired by ML and chatgpt, we came up with the idea to create our own music by detecting similar chord progressions among songs. For instance, we have observed some similarities between a traditional Chinese piece "House Racing" and the famous piano piece "Alla Turca" by Mozart.

The main idea of our project is to treat the chord progressions in each song as a subgraph (S) and create a joint graph (G) that contains all songs we are interested in. Then, with two community detection algorithm Girvan-Newman and Label Propagation, we will find communities C within G which will be the chord progression base for our song.

1.1. Representing music using graphs.

Definition 1. *A song graph $S(V, E)$ is a multigraph where V is the set of chords used in the music and E is the co-occurence of chords in the music.*

Definition 2. *A joint graph $G(V, E)$ is the union of song graphs*

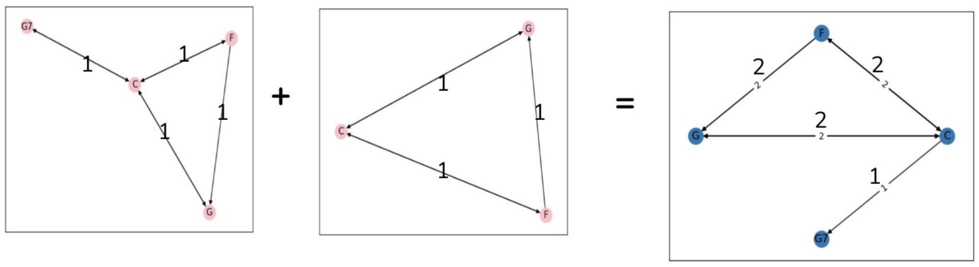


Figure 1. example combining song graph to joint graph

To create our joint graph, we use the following steps:
1. Initialize the joint graph: Create an empty directed graph that will become the joint graph.

2. Iterate through the song graphs: For each song graph, perform the following steps:

a. Add vertices: For each vertex in the song graph, check if it exists in the joint graph. If it doesn't, add the vertex to the joint graph.

b. Add edges and update weights: For each edge (u, v) in the song graph, check if the edge already exists in the joint graph. If it does, update the weight of the edge in the joint graph by incrementing its current weight by 1. If the edge doesn't exist in the joint graph, add the edge with an initial weight of 1.

3. Finalize the joint graph: At the end of this process, the joint graph will contain all the vertices and edges from the individual song graphs, preserving their directions. The edge weights in the joint graph will represent the number of song graphs containing that specific edge.

1.2. **Modularity and Community.** Before we dive into the algorithms of community detection in graphs, it is important to first establish a way to measure the quality of communities using the concept of Modularity. Modularity quantifies the degree to which a network can be divided into smaller, interconnected modules or communities, and serves as a metric for evaluating the quality of community structure within a complex network. A greater modularity indicates a stronger community structure.

The modularity Q of a partition of the graph can be calculated as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{1}$$

A_{ij}: A is the adjacency matrix, where A_{ij} is the weight of the edge between node i and node j. If there is no edge between i and j, $A_{ij} = 0$.

In calculating of (1),

- i and j iterate over all nodes in the network,
- d_i is the degree (sum of edge weights) of node i,
- $2m$ is the sum of all edge weights in the network,
- $\delta(c_i, c_j)$ is the Kronecker delta function, which equals 1 if $c_i = c_j$ and 0 otherwise.

Definition 3 (Community)**.** *A subset of vertices (nodes) within the graph such that connections between the nodes are denser than connections with the rest of the network.*

Community Detection algorithms can be divided to many categories. The most known method is Divisive methods, or top-down approaches, start with the entire network as a single community and iteratively split it into smaller communities based on certain criteria. Examples include the Girvan-Newman algorithm that this paper discusses.

Another method is Label propagation algorithms. These algorithms rely on the iterative propagation of labels through the network to identify communities. Nodes update their labels based on the labels of their neighbors, eventually reaching a consensus that reveals the community structure. Examples include the original Label Propagation Algorithm (LPA) and its variations, and this paper disucsses LPA.

There are also other approaches to this problem, including Clique-based methods, probability models, which we will not cover in this paper.

2. Girvan-Newman Algorithm

Many prior community detection algorithms, including Girvan-Newman, are based on the concept of "Network Centrality." It is a measure of the importance of a node (vertex) or an edge in a network (graph) construction. There are many (probably hundreds of) different types of centralities, although all are similar to the fundamental idea of others. Namely, the most common ones include:

Degree centrality (measures the number of edges that are connected to a node)

Betweenness centrality (the extent to which a node lies on the shortest paths between other pairs of nodes in the network)

Eigenvector centrality (a node's importance based on the importance of its neighbors)

PageRank centrality (measures a node's importance based on the number and quality of the links that point to it)

Network centrality measures can be useful for understanding the structure and dynamics of complex systems, such as social networks, biological networks, transportation networks, and many others. The Girvan-Newman community detection algorithm focuses on the concept of "edge betweeness."

Definition 4. *Edge Betweeness Centrality describes the frequency at which*

$$C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}} \tag{2}$$

σ_{st}*: number of shortest paths from node* s *to node (vertex)* t

$\sigma_{st}(e)$*: number of shortest paths from node* s *to node* t *that pass through the edge* e*.*

2.1. **Algorithm Description.**

By iteratively removing these central edges, the graph is gradually partitioned into smaller and smaller communities until each community is completely disconnected from the others. The detailed steps are as belowed:

(1) Calculate the betweenness centrality of all edges in the network.
(2) Remove the edge(s) with the highest betweenness centrality value(s).
(3) Recalculate the betweenness centrality of all edges in the remaining network.
(4) Repeat steps 2 and 3 until a desired number of communities is obtained or until the network breaks down into disconnected components.

Notice that if a graph $G(V, E)$ has a cut-edge, then the cut-edge is more likely to 1) has a higher edge-betweeness and 2) belong to none of the community. Here is a simple demonstration. (fig) In the graph below, we have 3 cut-edges (2,7), (9,12), (4,5).

Iterating the Girvan-Newman algorithm twice, that is, removing the edge with the highest edge-betweeness in the remaining graph twice, leads to three communities above. As expected, we have cut-edges (2,7) and (9,12) being removed first.

2.2. **Performance and Limitations.**

The Girvan-Newman algorithm has been used in a wide range of applications, from social network analysis to bioinformatics. It is a powerful tool for identifying the underlying structure of complex networks and understanding the relationships between different parts of the network. However, being the very first algorithm in community detection, it has some limitations that can not be ignored. The two major problems are efficiency and resolution limit concerns.

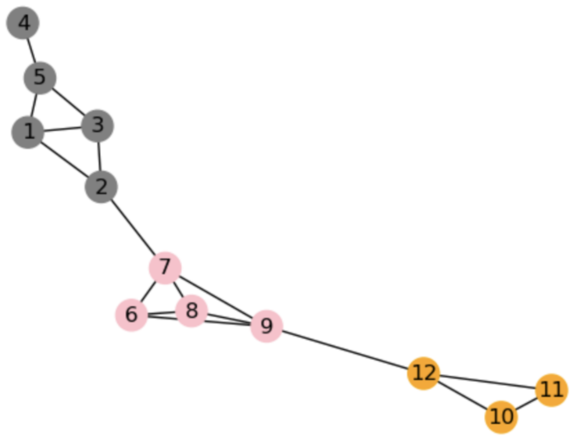


FIGURE 2. example network with cut-edge

(1) Efficiency concern: The algorithm can become computationally expensive for large graphs. The algorithm has to compute the betweenness centrality of all edges in the graph, which can be time-consuming and memory-intensive.
(2) Resolution limit concern: The algorithm has a resolution limit, which means that it cannot detect communities that are smaller than a certain size. This is because the algorithm works by removing edges with high betweenness centrality, which tends to break up large communities into smaller ones.

Notice that our initial construction of a joint graph as a weighted and directed multigraph, Girvan-Newman algorithm fails to handle communities with different internal structures because it only detects communities based on the edge betweenness centrality metric, which does not take into account the orientation or weight of an edge.

2.3. **Code Implementation.** Here we show the implementation of Girvan-Newman in Python. This demonstration is consistent with figure 1.

```python
import networkx as nx
G = nx.Graph()
# define a graph. Here, G is a simple unweighted, undirected graph
G.add_edges_from([[1,2], [2,3], [1,3], [4,5], [1,5], [3,5],
                  [6,7], [7,8], [8,9], [6,9], [6,8], [7,9],
                  [10,11], [11,12], [10,12],
                  [9,12], [2,7]])
def edge_to_remove(graph):
    G_dict = nx.edge_betweenness_centrality(graph)
    edge = ()

    # extract the edge with highest edge betweenness centrality score
    for key, value in sorted(G_dict.items(), key=lambda item: item[1], reverse=True)
        ↪ :
        edge = key
        break
    return edge
```

```python
def girvan_newman(graph):
    # find number of connected components
    sg = nx.connected_components(graph)
    sg_count = nx.number_connected_components(graph)
    while sg_count <= 2:
        graph.remove_edge(edge_to_remove(graph)[0], edge_to_remove(graph)[1])
        sg = nx.connected_components(graph)
        sg_count = nx.number_connected_components(graph)
    return sg
# find communities in the graph
c = girvan_newman(G.copy())

# find the nodes forming the communities
node_groups = []
for i in c: node_groups.append(list(i))
```

3. Label Propagation

The Label Propagation algorithm (LPA) is a fast and efficient community detection method that can uncover communities in large-scale networks. The algorithm is based on the idea that densely connected groups of nodes should share the same community label. LPA operates by iteratively updating the community label of each node based on the labels of its neighbors, eventually reaching a consensus where the labels stabilize.

3.1. **Algorithm Description.** The Label Propagation algorithm can be summarized as follows:

(1) Initialize the algorithm by assigning a unique label to each node in the network, i.e., for each node i, set $l_i(0) = i$, where $l_i(t)$ denotes the label of node i at iteration t.
(2) For each iteration $t > 0$, update the label of each node i based on the labels of its neighbors during the previous iteration, $l_j(t-1)$ for each neighbor j of node i. Set $l_i(t)$ to the label that occurs most frequently among its neighbors. If there are multiple labels with the same frequency, choose one of them randomly.
(3) Continue updating the labels until convergence is reached, which occurs when the labels remain unchanged between consecutive iterations or a predefined stopping criterion is met.
(4) After convergence, nodes with the same label belong to the same community.

3.2. **Code Implementation.** Here we show the implementation of Label Propogation in Python.

```python
import networkx as nx
import random

def label_propagation(G):
    labels = {node: node for node in G.nodes()}
    nodes = list(G.nodes())

    while True:
        random.shuffle(nodes)
        prev_labels = labels.copy()

        for node in nodes:
            neighbor_labels = [labels[n] for n in G.neighbors(node)]
            labels[node] = max(neighbor_labels, key=neighbor_labels.count)

        if prev_labels == labels:
            break

    communities = {}
    for node, label in labels.items():
        if label not in communities:
            communities[label] = []
        communities[label].append(node)

    return communities.values()
```

3.3. **Performance and Limitations.** The Label Propagation algorithm has a time complexity of $O(m)$, where m is the number of edges in the network. This makes it particularly suitable for large-scale networks, as the algorithm can efficiently uncover communities with a low computational cost.

However, LPA has some limitations. It can be sensitive to the initial node ordering and may produce different community structures in different runs due to its random selection of labels when there are ties. Furthermore, the algorithm may not always converge to a clear partition of the network, and in some cases, oscillations between different label configurations can occur.

Despite these limitations, the Label Propagation algorithm remains a valuable tool in community detection due to its simplicity, efficiency, and scalability.

4. Algorithm Comparison

Label propagation algorithm is often considered more efficient than Girvan-Newman algorithm (as shwon in figure 3, an empirical experiment) because it requires fewer computations to reach a community detection solution. Girvan-Newman algorithm involves repeatedly calculating edge betweenness centrality and removing the edges with the highest values, which can be computationally expensive for large graphs. On the other hand, label propagation algorithm iteratively updates the community labels of the nodes based on the labels of their neighbors. The algorithm converges when the community labels no longer change.

However, the label propagation algorithm is not consistent. While Girvan-Newman algorithm is consistent because the edge betweeness stays fixed for a graph; Label propagation algorithm may produce different community detection outcome depending on the seed (the vertex start with) and the initial color assignment. There is another layer of uncertainty in the outcome of Label propagation algorithm introduced by the use of ”random walk.”

5. Application Demonstration

Notice that the joint graph we have is a weighted and directed multigraph as out joint graph (union of all song graphs). However, the first algorithm, Girvan-Newman is generally used for undirected and unweighted simple graph; while Label propogation can be used to detect communities in weighted graph (by re-weighting the adjacency matrix) but is not designed to deal with directed graph. Thus, the question of ”How can we best detected a community within a directed graph” naturally rises. After going over literature, we found that the problem of detecting communities

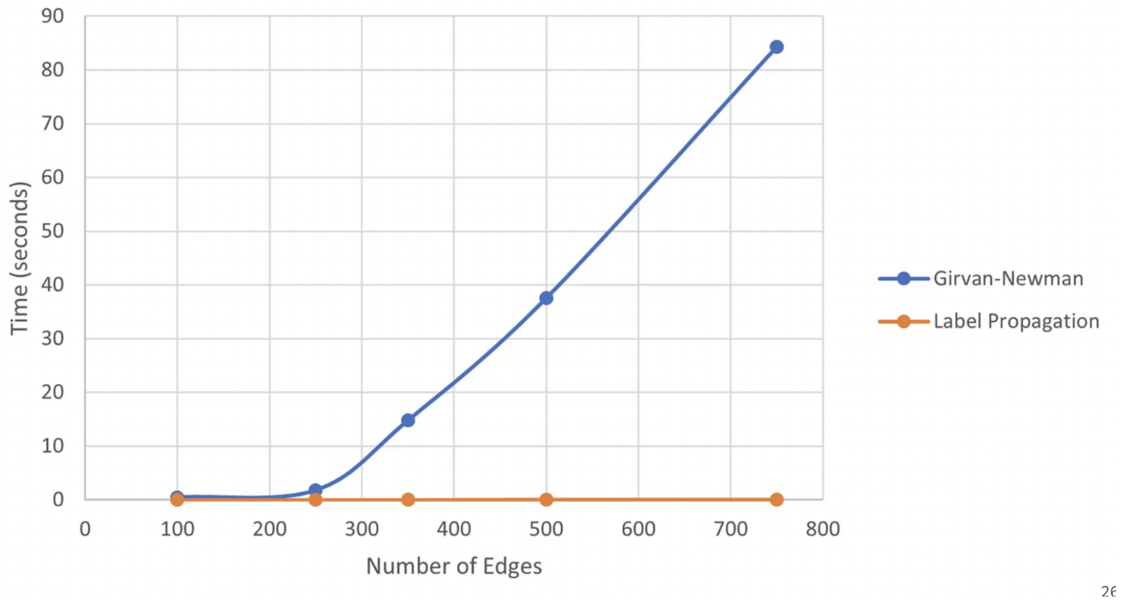
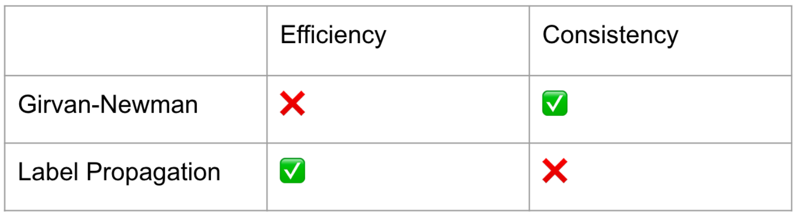


FIGURE 3. algorithm efficiency comparison



	Efficiency	Consistency
Girvan-Newman	❌	✅
Label Propagation	✅	❌

FIGURE 4. algorithm comparison

in a directed network is much more challenging that that in the undirected case. In a directed network, the connected relationship between two vertices is asymmetric. Thus, the definition of "Centrality," "Modularity" no longer works. Another challenge with directed networks is that they can have multiple types of communities, depending on whether we consider only the incoming links or the outgoing links. This means that we need to choose which type of community we are interested in detecting, and this can affect the results we obtain.

Thus, we proposed a steps to follow for solving our problem, which can be implemented in the future:

5.1. **Possible steps to follow for detecting communities for chord progression in a joint graph.**

(1) Get n communities with Girvan-Newman Algorithm (since consistent).
(2) Get m communities within each n communities with Label Propagation Algorithm (since dealing with weight; more efficient).
(3) Refer to the original graph for direction information.
(4) Define direction within each community:
  - $\rightarrow$ and $\leftarrow$: if proportion of $\rightarrow$ on an edge is between 40-60 percent.

- $\rightarrow$: if the proportion of $\rightarrow$ on an edge > 60 percent.
- $\leftarrow$: if the proportion of $\leftarrow$ on an edge > 60 percent.

Note that the notation $\rightarrow$ and $\leftarrow$ represent the direction of edges in the graph.

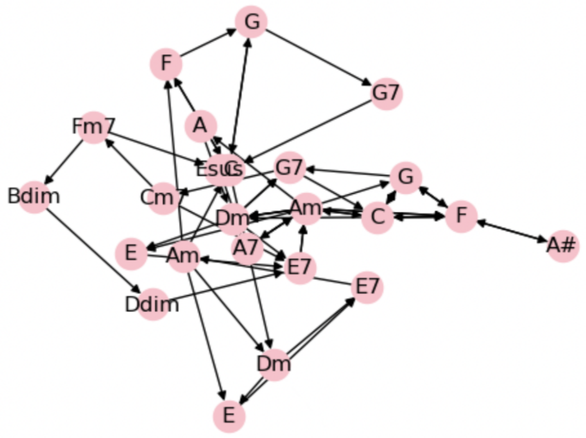


FIGURE 5. joint graph of four songs

For this paper, we only considered a simple unweighted and undirected graph for demenstration purpose. Here is a demonstration of the use of two algorithms in the same undirected unweighted joint graph generated using four song graphs. The four songs we chose are Happy Birthday Song, Fly me to the moon, Baby ft. Ludacris, Waltz in C major by Olga Scheps. Notice that we "standardize" all song's tonality to C when fetching the chords for each song graph. Then, we produced a joint graph for all chord progrssions used in the song graphs. (directed and weighted, weights are stored in a separate array) We eliminate directions and weight and use two algorithms to detect the communities. Vertices in the same color belong to the same community. The community with C major chord is circled.

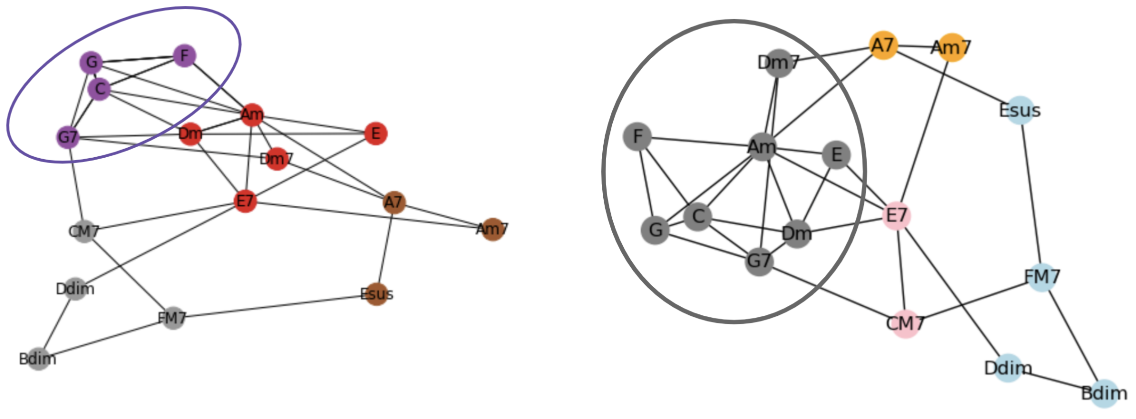


FIGURE 6. detection result, left: label propagation; right: Girvan-Newman

References

[1] Zhu, X., & amp; Rogers, T. T. (2012). A cognitive study of learning with labeled and unlabeled data.

[2] Newman, M. E. (2004). Detecting community structure in networks. The European Physical Journal B - Condensed Matter, 38(2), 321–330.

[3] Fortunato, S. (2010). Community detection in graphs. Physics Reports, 486(3-5), 75–174.

[4] Shalev Itzkovitz, S., Milo, R., Kashtan, N., Levitt, R., Lahav, A., & Alon, U. (2006). Recurring harmonic walks and network motifs in western music.

[5] Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. Physical Review E, 69(2).

[6] Clauset, A., Newman, M. E. J., & Moore, C. (2004). Finding community structure in very large networks. Physical Review E, 70(6).

[7] Yao, Q., Wu, Y., Huang, T., Dong, Y., Xu, B., Liu, Y., Huang, Z. (2020). Modeling the Stock Relation with Graph Networks for Overnight Stock Movement Prediction. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI-20)* (pp. 4524-4530).

[8] Cheng, Q., Yang, W., Wang, Q., Zhang, Y., Zhang, Z. (2020). Knowledge Graph-based Event Embedding Framework for Financial Quantitative Investments. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 2497-2500).

[9] Di Battista, G., Eades, P., Tamassia, R. (2017). Analysis of Equity Markets: A Graph Theory Approach. *SIAM Undergraduate Research Online*, 10, 539-552.

[10] Wang, X., Chen, X. (2020). Stock Network Stability After Crashes Based on Entropy Method. *Frontiers in Physics*, 8, 163.