Report for Lab activity: analysis of music networks

Marino Oliveros Blanco NIU:1668563
Pere Mayol Carbonell NIU:1669503

Important notes: in the code both client secret and client id were removed as well as the private path/directory for the graph and .csv files created and used, these were simplified /Users/example/example_folder/gB.graphml -> gB.graphml.

1. Introduction

In this report, we will explain how we have faced the Lab activities. In these exercises, we have explored the capabilities of the spotipy libraries and Spotify's API, connecting them to our code and completing our objectives. This is separated in the acquisition and preprocessing of the data, and afterward analyzing and visualizing it. Here we will explain the problems and setbacks we found along the way, as well as how our code works.

2. Part 1: Data acquisition

For acquiring the data, we will use the client object specific for our connection to our spotify app. Through this, we can access the artists and their information, including their neighbors, connections and personal creations.

There are three functions in this part.
The first one {`def search_artist`} simply gets the client object to connect to and a name of an artist[str] to get their ID using the function search in the spotipy library.

The second one {`def crawler`} is the one we will use to search through the found items in neighboring artists. This initializes a queue for BFS or a stack for DFS, where we will put the artists found.
Afterward, everything happens inside a while that stops when we reach the maximum number of nodes we want to crawl. They are both initialized using the seed and an empty value. What we do inside is, basically, first pop the oldest item in the stack or queue and get them in a variable called current_artist_id and the parent_id. Moreover, we add the artist we currently are in to the visited list, to have them all in one place and also control the amount of elements we visit. We then get the information of the artist, using '`sp.artisd'` and add all the information to a node along with their name. If there is a parent_id found in the queue or stack, then we also add an edge to connect them.
Then, we get the neighbors of this node through the function '`artist_related_artists`' in the spotipy library, which returns a dictionary with information on every item related to the aforementioned artist. We then create a node for the information regarding the neighboring artists, add the actual and neighbor node to the all_related_artist list and add to the queue or stack the ID to the related artist and to its original parent. Afterward, we create the edges for all related artists and convert it to a format we can work with
[`nx.write_graphml(G,out_filename)`] This is what is returned from the function.

Throughout all of these, we set up a counter to check how many API calls we make along the way and also make an exception so that if an error pops out for calling the API too much, we are informed and can update the code or

The last function in this part {`def get_track_data`} gets the graphs that result of our findings in the crawler function and creates a dataset. To do this, we have implemented two main tools, the get_track_data per se and also an inside function that checks if the API calls can be made for the top tracks and audio feature functions. This second function checks if the call can be made or if there is a 429 error meaning the API has collapsed due to too many calls. To neutralize this option, we have applied a delay between calls and do it in batches of artists and not one by one.
The main functionality of the function works as follows: it visits all nodes in the graphs provided and gets the needed information on the top five tracks of the artists found. This dictionary with everything is then converted to a pandas DataFrame and download it as a CSV.

All of this is implemented together as a hole in the main function, where we can visualize the created graphs, the dataset and see which are the properties of the nodes inside of them, along with another function {`def degree_statistics`} which is basically used to get the degree's information for each node.

2.1. Questions to answer [Part 1: Data acquisition]

2.1.1. Regarding the graphs Gb[BFS] and Gd[DFS]:.

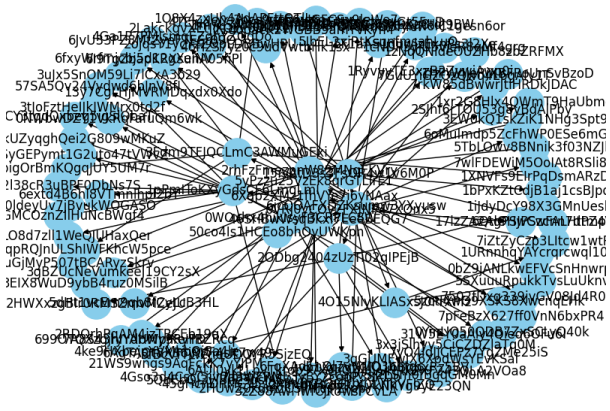
They are created using the same inputs except for the search algorithm that they use to crawl through the graph. The main difference between them is then going to be that the storage of artists in Gb is going to be in a queue and for Gd a stack. So, for the graph using the BFS search, we see the structure tends to be wider and respect more levels. The second one, shows a deeper and more linear structure, reflecting the depth-first nature. As a result, even if both algorithms explore the same number of nodes, the actual nodes visited and the connections between them can differ significantly, leading to a difference in the graph order.

1. Order and size.

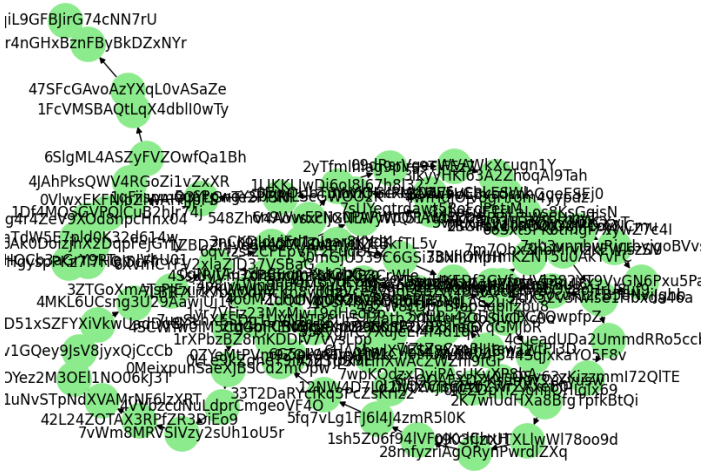
- For the Graph Gb[BFS], the stats are: {'number_of_nodes': 473, 'number_of_edges': 2000}
- For the Graph Gd[DFS], the stats are: {'number_of_nodes': 601, 'number_of_edges': 1995}

They have different values because BFS explores nodes level by level, which can lead to discovering more immediate neighbors, whereas DFS explores as deep as possible before backtracking. In a case where the graph has some dense immediate neighbors, then the BFS would discover more nodes, but in the case of the DFS, by going deep into one branch before backtracking, DFS might not create as many edges within the local neighborhood, instead forming longer chains of nodes. This can result in fewer edges overall because DFS often connects nodes in a linear or hierarchical fashion, with fewer cross-connections.

BFS:



DFS:



2.1.2. Indicate the minimum, maximum, and median of the in-degree and out-degree of the two graphs (gB and gD). Justify the obtained values.

Using the function mentioned before for the degree statistics, we get the following statistics for the graphs Gb and Gd:

a. For the Graph Gb[BFS], the stats are: {'in_degree': {'min': 1, 'max': 38, 'median': 2}, 'out_degree': {'min': 0, 'max': 20, 'median': 0}, 'number_of_nodes': 473, 'number_of_edges': 2000}
b. For the Graph Gd[DFS], the stats are: {'in_degree': {'min': 0, 'max': 33, 'median': 2}, 'out_degree': {'min': 0, 'max': 20, 'median': 0}, 'number_of_nodes': 601, 'number_of_edges': 1995}

Given the characteristics mentioned before and the visualizations of the graphs, we can see these goes hand in hand what we could expect. The in degrees vary since they discover nodes differently and BFS does it exploring more close by neighbors, this is why we do not find any node with a value of 0 in that scope, because BFS tends to connect nodes more within the same level, ensuring higher in-degrees for intermediate nodes. On the other hand, DFS

explores deep branches, so it makes sense that we end up with more isolated nodes.
The out degree values are similar because at the end both algorithms ensure connecting the many neighbors.

2.1.3. Indicate the number of songs in the dataset D and the number of different artists and albums that appear in it

Songs: 1269
This amount of songs makes sense taking into account we are taking the intersection of the graphs and then take their top songs. Therefore, it makes complete sense, the intersection leaves us with 138 artists, and we get their top for each of them.
Albums: 733
Obviously, the number of albums is smaller, since some of the albums in the top tracks for each artist will be from the same one.
Artists: 138

Lab AGX 202324 Report Part 2: data preprocessing

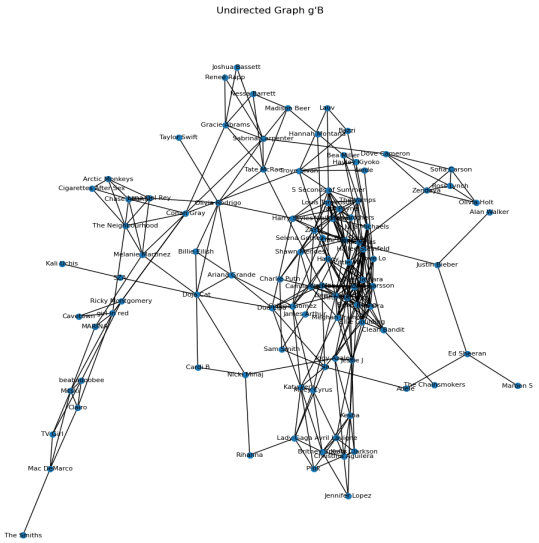
1. **(1.5 points) Justify whether the directed graphs obtained from the initial exploration of the crawler (gB and gD) can have more than one weakly connected component and strongly connected component, and explain why. Indicate the relationship with the selection of a single seed.**

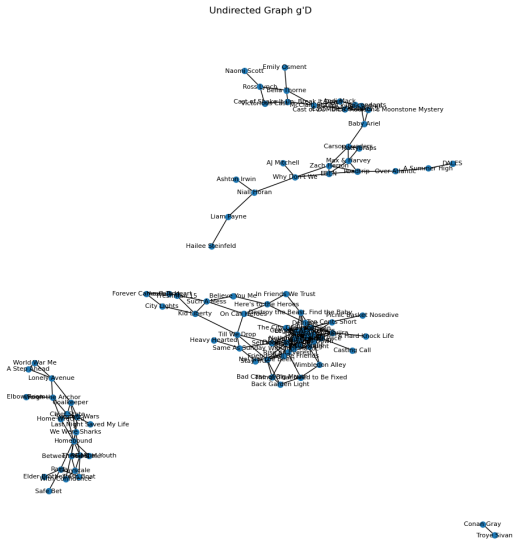
When a single seed is used to start the crawling process, the resulting graph will typically consist of one strongly connected component and potentially multiple weakly connected components. This is because the crawler can only follow outgoing links from the initial seed and the nodes it discovers, but it cannot discover nodes or links that are not reachable from the seed.

The strongly connected component represents the subgraph where all nodes are reachable from each other through directed paths. However, there may exist other subgraphs that are reachable from the strongly connected component but not vice versa, forming weakly connected components.

Therefore, the selection of a single seed can result in multiple weakly connected components in the directed graphs gB and gD, as the crawler may not be able to discover all the nodes and links in the network due to the limitations of the crawling process.

As we can see by the outputs gBp has 1 connected component and gDp has 4.





2. **(0.5 points) Can the number of connected components in the undirected graphs (gB′ and gD′) be higher than the number of weakly connected components of its respective directed graph (gB and gD)? Provide a minimal example to showcase your answer.**

In an undirected graph, two nodes are considered connected if there exists a path between them, regardless of the direction of the edges. On the other hand, in a directed graph, two nodes are weakly connected if there exists a path between them, ignoring the direction of the edges.

Consider the following example:

Directed graph: A -> B B -> C C -> D D -> A

In the directed graph, there is only one weakly connected component, as all nodes are reachable from each other if the direction of the edges is ignored.

Undirected graph: A - B B - C C - D

In the undirected graph, there are two connected components: {A, B} and {C, D}. This is because the edge between D and A in the directed graph has been removed in the undirected graph, breaking the connectivity between the two components.

Therefore, by converting a directed graph to an undirected graph, the number of connected components can increase if certain edges are not bidirectional in the original directed graph. This is the case in the example provided, where the undirected graph has more connected components than the weakly connected components of the corresponding directed graph.

3. **(1 point) Generate a preliminary report from the undirected graph with weights (gw).**

Due to the high number of similarities between artists we would try to implement some sort of normalization to see our results better (this is done in the notebook in our github, not on the skeleton.py file).

   (a) Which are the two most (respectively, least) similar artists? What graph attribute allows you to answer this question?

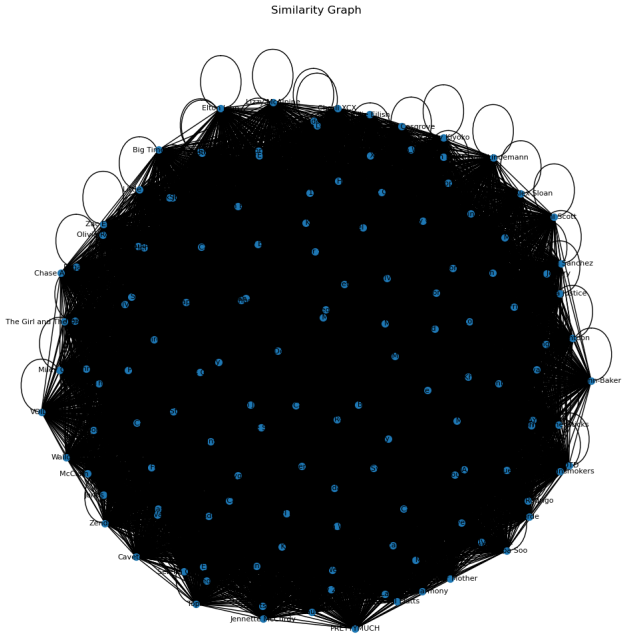
The two most similar artists are found by sorting the edges of the similarity graph by weight in descending order, and taking the first edge, excluding any self-loops. This is done in the find_most_least_similar_artists function the most similar artists are Lana del Rey and Taylor Swift the same with the least similar ones which are Hannah Montana and Mitchel Musso.

The graph attribute that allows answering this is the edge weights in the similarity graph, which represent the similarity between each pair of artists.

(b) Which is the artist most (and least) similar to all the other artists in the network? What graph attribute allows you to answer this question?

The artist most (and least) similar to all other artists is found by calculating the average similarity of each node to its neighbors, and taking the node with the highest (lowest) average similarity. This is done in the find_most_least_similar_to_all function. The graph attribute that allows answering this is the edge weights in the similarity graph, combined with the degree of each node, which allows computing the average similarity of a node to its neighbors.

The most similar artists to all others are Hailee Steinfeld and the least Phillipa Soo.



Part 3: Analysis

Lab AGX 202324 P3 Report: Data Analysis

1. **(0.75 points) Study the number of common nodes between the obtained graphs. Use the function num common nodes.**

Using the function num_common_nodes we obtain the following.

(a) How many nodes are shared between gB and gD? What information does this tell us about the importance of the algorithm used by the crawler (i.e. the scheduler) to decide next nodes to crawl?

Number of common nodes between gB and gD: 138

This indicates that the "Related Artists" algorithm on Spotify has a different focus or criteria for determining artist relationships.

(b) How many nodes are shared between gB and gB′ ? What information does this tell us about the reciprocity of gB? And about Spotify's artist related algorithm?

Number of common nodes between gB and gBp: 99

The lower number of common nodes (99) between gB and gBp compared to the total number of nodes in gB suggests that the "Related Artists" relationships on Spotify are not completely reciprocal. If the relationships were fully reciprocal, we would expect gBp (the reciprocal version of gB) to have the same set of nodes as gB

The relatively low overlap between gB and gBp also indicates that Spotify's "Related Artists" algorithm may have a different focus or criteria compared to a simple reciprocation of relationships. It suggests that the algorithm likely incorporates additional factors or objectives beyond just mirroring the relationships in both directions.

2. **(0.5 points) Calculate the 25 most central nodes in the graph gB′ using both degree centrality and betweenness centrality. How many nodes are there in common between the two sets? Explain what information this gives us about the analyzed graph.**

Number of common nodes between degree and betweenness centrality: 9.

The overlap between the top nodes ranked by degree centrality and betweenness centrality in the graph gB′ highlights the existence of highly influential and strategically positioned nodes that are crucial for maintaining connectivity, facilitating information diffusion, and ensuring the network's robustness and resilience.

3. **(0.5 points) Find cliques of size greater than or equal to min size clique in the graphs gB′ and gD′ . The value of the variable min size clique will depend on the graph. Choose the maximum value that generates at least 2 cliques. Indicate the value you chose for min size clique and the total number of cliques you found for each size. Calculate and indicate the total number of different nodes that are part of all these cliques and compare the results from the two graphs.**

The maximum value that generates at least 2 cliques (in total from gBp and gDp) is 7; this was obtained by trial and from 4-7 until 8 did not provide 2 cliques in total.

Size 7 generates 4 cliques in gBp with a total number of nodes in the cliques of 18 and 1 clique in gDp with a total of 7 nodes in cliques.

4. **(0.5 points) Choose one of the cliques with the maximum size and analyze the artists that are part of it. Try to find some characteristic that defines these artists and explain it.**

The largest clique in gBp is composed by the following: [('64M6ah0SkkRsnPGtGiRAbb', 'Bebe Rexha'), ('1Xylc3o4UrD53lo9CvFvVg', 'Zara Larsson'), ('1zNqDE7qDGCsyzJwohVaoX', 'Anne-Marie'), ('5CCwRZC6euC8Odo6y9X8jr', 'Rita Ora'), ('2wUjUUtkb5lvLKcGKsKqsR', 'Alessia Cara'), ('5p7f24Rk5HkUZsaS3BLG5F', 'Hailee Steinfeld'), ('1l8Fu6IkuTP0U5QetQJ5Xt', 'Fifth Harmony'), ('3e7awlrlDSwF3iM0WBjGMp', 'Little Mix')]

These artists are primarily female pop and R&B which reached their peak of fame in the 2010s to a similar demographic.

5. **(0.5 points) Detects communities in the graph gD. Explain which algorithm and parameters you used, and what is the modularity of the obtained partitioning. Do you consider the partitioning to be good?**

The code uses the Louvain method to detect communities in the graph gD. The detect_communities function takes the graph and the method name as input. When the method is set to 'louvain', it converts the directed graph to an undirected graph (if necessary) and then applies the community_louvain.best_partition function from the community module to find the partition of nodes into communities.

The modularity of the obtained partitioning is 0.7218587984529132. A modularity value closer to 1 indicates better community structure in the network. A modularity value of 0.7218587984529132 is considered relatively high, suggesting that the partitioning is reasonably good, with a clear community structure in the gD graph. By our slides >0.3 is considered a sign of community structure.

6. **(1 point) Suppose that Spotify recommends artists based on the graphs obtained by the crawler (gB or gD). While a user is listening to a song by an artist, the player will randomly select a recommended artist (from the successors of the currently listened artist in the graph) and add a song by that artist to the playback queue.**

   (a) Suppose you want to launch an advertising campaign through Spotify. Spotify allows playing advertisements when listening to music by a specific artist. To do this, you have to pay 100 euros for each artist to which you want to add ads. What is the minimum cost you have to pay to ensure that a user who listens to music infinitely will hear your ad at some point? The user can start listening to music by any artist (belonging to the obtained graphs). Provide the costs for the graphs gB and gD, and justify your answer.

The minimum cost to ensure that a user who listens to music infinitely will hear an ad at some point is determined by the number of strongly connected components (SCCs) in the graph. Each SCC needs at least one artist to have an ad placed to ensure that a user traversing that component will eventually encounter the ad.

The min_ad_cost function finds all strongly connected components using nx.strongly_connected_components and returns the number of SCCs as the minimum cost.

For graph gB, the minimum cost is 37400 euros, as there are 374 strongly connected components. For graph gD, the minimum cost is 50600 euros, as there are 506 strongly connected components.

   (b) Suppose you only have 400 euros for advertising. Which selection of artists ensures a better spread of your ad? Indicate the selected artists and explain the reason for the selection for the graphs gB and gD.

With a budget of 400 euros, the best strategy is to select the most central artists within each strongly connected component, as they have the highest likelihood of being reached by users traversing the component.

The select_artists_for_budget function implements this strategy:

1. It finds all strongly connected components in the graph.
2. For each SCC, it finds the node with the highest degree centrality, considering the subgraph induced by the SCC.
3. It sorts the central nodes by their centrality in descending order.
4. It selects the top central nodes, subject to the budget constraint (assuming each artist costs 100 euros).

For graph gB, the selected artists within the 400 euros budget are:
[('6oW9KRAZbC1xOlmg2RRyFL', 'McClain Sisters'), ('6G9bygHlCyPgNGxK2l3YdE', 'Vanessa Hudgens'), ('5bmqhxWk9SEFDGIzWpSjVJ', 'THE DRIVER ERA'), ('542yUd4rGzUEOLd1diV94f', 'Rocky')]

For graph gD, the selected artists within the 400 euros budget are:
[('4Uc8Dsxct0oMqx0P6i60ea', 'Conan Gray'), ('1gzqMaNtHI85YIVvZlcZHe', 'Nothing Planned'), ('3dUtEOe21FQJhD834hUkAm', 'Brigades'), ('3XHn51FdwX1GZCGMz6RMYM', 'Pentimento')]

7. **(1 point) Consider a recommendation model similar to the previous one, in which the player shows the user a set of other artists (defined by the successors of the currently listened artist in the graph), and the user can choose which artist to listen to from that set. Assume that users are familiar with the recommendation graph, and in this case, the gB graph is always used.**

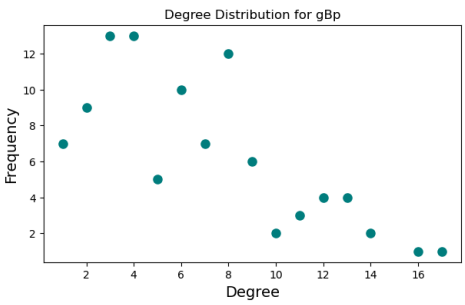
   (a) If you start by listening to the artist Taylor Swift and your favorite artist is THE DRIVER ERA, how many hops will you need at minimum to reach it? Give an example of the artists you would have to listen to in order to reach it.

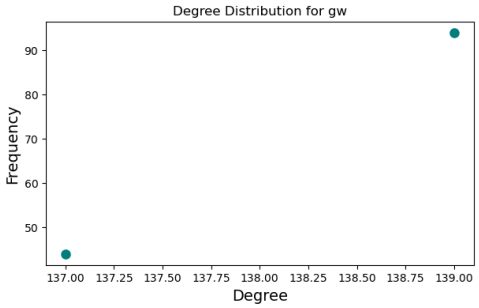
The minimum number of hops is 3. An example hop path would be : [('06HL4z0CvFAxyc27GXpf02', 'Taylor Swift'), ('1McMsnEElThX1knmY4oliG', 'Olivia Rodrigo'), ('4VdV2qRAYBLINR6uU72V1J', 'Joshua Bassett'), ('5bmqhxWk9SEFDGIzWpSjVJ', 'THE DRIVER ERA')] making 3 hops in total.

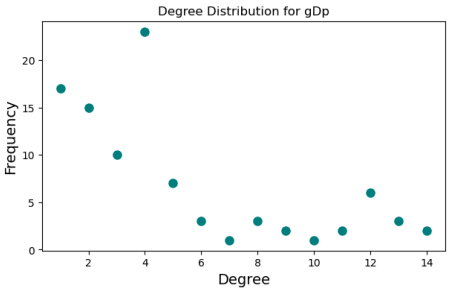
Part 4: Visualization

Lab AGX 202324 P4 Report: Data Visualization.

(a) What are the degree distributions of the three obtained undirected graphs like?



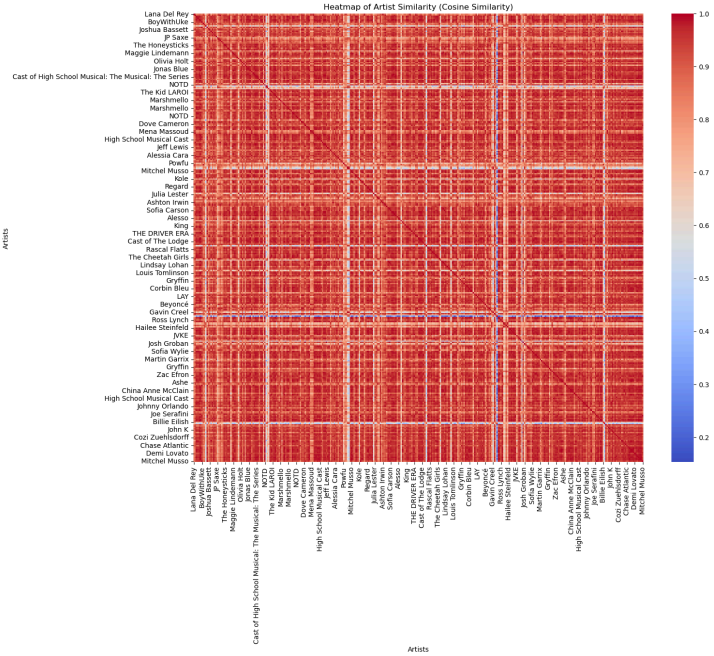




Here we can see the three plots for the graph's degree distribution. For the gDp and gBp graphs, the values are more scattered and smaller. On the other hand, for gw, it is clear that it is a very dense and compact graph since all nodes have a degree of either 139 or 137.

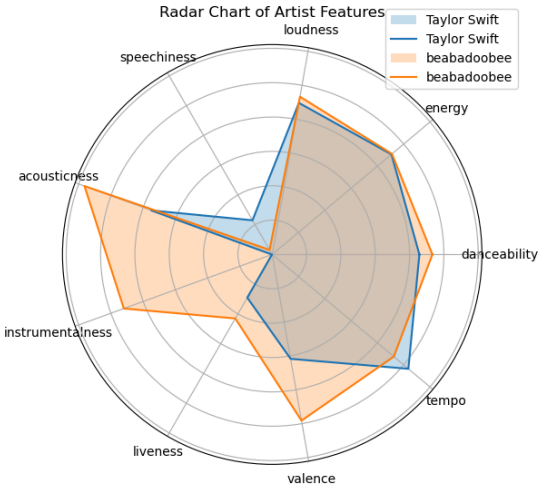
For gDp specifically, we can see as the degree gets higher, the number of nodes that pertain to that section lowers almost every time follows a perfect drop except for the nodes with degree 4 which create a spike.
For gBp, the values do not follow the same path as in the latter graph since we can see as the degree is higher, eventually there is the drop, but for most of the representation the values seem arbitrary. Nevertheless, we can see the majority of nodes have lower degrees.

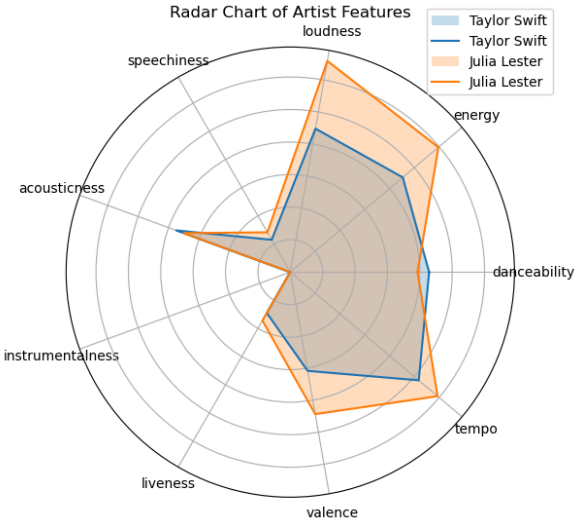
(b) What can you infer from the similarity heatmap regarding the algorithm that selects related artists on Spotify?



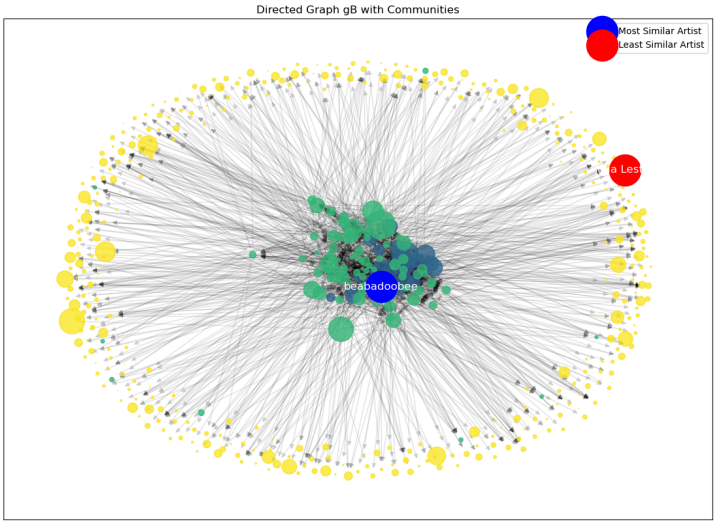
As we can see, most artist have a high correlation and similarity value between them. Taking a look at the colors and the guide, overall redness indicates that these artists are indeed related.
There are some artists that have lower values overall, as we can see the blueish lines that cross the plot, but they do not have worryingly low values.
This is just the case for two artists. That have a very low value with respect to almost everyone.

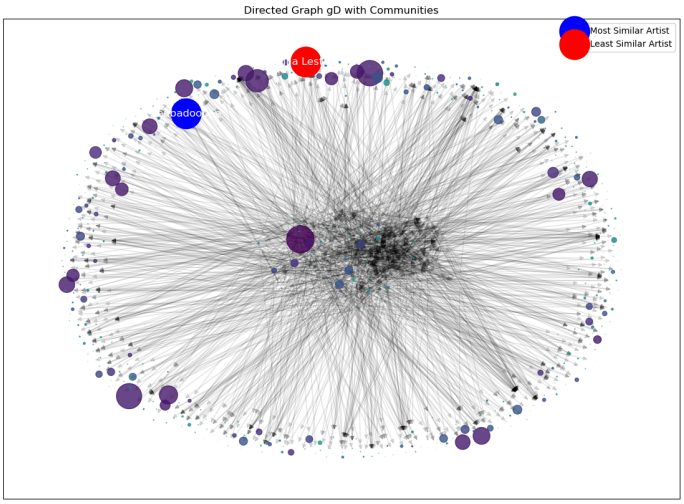
(c) Is there any relationship between the similarity of artists obtained from their audio features and the distances of the artists in the directed graphs? For instance, consider Taylor Swift and her most and least similar artists as determined in exercises 4.b and 4.c.





The plots for these artists are the following, on the left is the most related in gw and on the right the least related. So for both of them we have plotted the distance to Taylor in the directed graphs and here is what we got:





As we can see in the gD, the distance between these nodes is similar, and we cannot see any major difference. Nevertheless, in the gB we can clearly see that the most similar artist is much closer to the center and the least similar is in the periphery of the graph.

(d) At which percentile would you prune the edges of the weighted similarity graph gw to ensure the size of the largest connected component is preserved while minimizing the amount of edges in the graph?

Inspecting what the changes are, depending on the percentile, we can see the changes for each input and check when we can preserve the biggest size for the largest connected component and minimizing the amount of edges. We can see that the number of nodes in the largest connected component stays the same through 60-70 percentile:

For 30 ->{'degree': {'min': 4, 'max': 123, 'median': 111}, 'number_of_nodes': 137, 'number_of_edges': 6685}
For 62-->{'degree': {'min': 1, 'max': 80, 'median': 64}, 'number_of_nodes': 137, 'number_of_edges': 3628}
For 70 -> {'degree': {'min': 2, 'max': 69, 'median': 48}, 'number_of_nodes': 135, 'number_of_edges': 2863}
For 86-> {'degree': {'min': 1, 'max': 42, 'median': 21}, 'number_of_nodes': 130, 'number_of_edges': 1337}
For 95-> {'degree': {'min': 1, 'max': 20, 'median': 8}, 'number_of_nodes': 113, 'number_of_edges': 478}

The best option would be to pick a high percentile, since the number of nodes does not decrease that much for the largest connected component and the number of edges does decrease pretty drastically at the end. I would pick any value between 80 and 90, where we still have 132-129 components or so and the edges go down by almost half compared to the 70 percentile.