Graph-based Music Recommendation System

Aniruddha Murali, Austin Peng, Devan Moses, Heidi Yap, Pranav Manjunath*
Team-006

1 INTRODUCTION

The usage of recommendation systems has recently become widespread due to the increasing amount of usable online data. Consumer-based services (ex. Amazon, Netflix, Spotify, etc.) need methods of quickly parsing large amounts of data and presenting relevant information to their users. Our research focuses on music recommendation systems and aims to better personalize with each user and recommend songs accordingly.

2 PROBLEM DEFINITION

Simply put, the problem is to develop an interactive visualization tool that can help users better understand how songs are recommended to them.

More specifically, our goal is to implement a music recommendation system accompanied by a visualization using force-directed graphs. The system will provide users with personalized song recommendations based on their existing song preferences. The visualization tool will display these recommendations in an intuitive, easily understandable, and aesthetic way.

3 LITERATURE SURVEY

Current music recommender systems use collaborative filtering, content-based filtering, demographic filtering, knowledge-based filtering (using knowledge graphs), metadata-based models, hybrid models, and more [3, 4, 9]. Hybrid models generally incorporate a mix of different techniques. Others even use embedding techniques with knowledge graphs to enhance performance [6, 13]. In our research, we will focus on the knowledge graph-based method.

A graph/hypergraph-based model can be used to represent heterogeneous data sources in music recommendation systems (ex. user, artist, releases, tags, and more). The connections within the graph can be used by algorithms to compute a ranking of songs for a given user. Most existing music recommendation systems suffer from data sparsity which a hypergraph model can address through integrating different data types (ex. user listening histories, music features, and social network information) into a single model [2]. However, [2] limited evaluation metrics (precision, recall, and F1-score) when they could have considered user satisfaction or diversity of recommendations.

Music genes (ex. melody, rhythm, tempo) and social genes (ex. mood, region, language) can be exploited to increase hybrid recommendation system performance [12]. For instance, predicted mood from song lyrics have been used to recommend songs [10]. Machine learning models can be trained with music audio to accurately recommend songs by predicting latent factors such as music and social genes from songs [1]. It is also possible to form a policy through training an agent on the graph with beam search (algorithm that can make accurate recommendations with small amounts of user preference info) [11]. However, these forms of data are either hard to obtain or too difficult to classify with high accuracy in a short time frame.

A system using hypergraph embeddings can be utilized to infer user-song similarities through vector mapping [6]. When combined with music genes, a hypergraph can be leveraged to further increase recommendation performance. In addition, the authors of [6] mentioned that time-based information and clustering of users (ex. community detection) are potential methods that can further improve their methodology. A downside to this approach is the need to constantly retrain the embeddings.

Other features to consider are user preferences, item novelty, and music track quality. Constructing a preference directed graph to measure attractiveness of a song and a positive correlation undirected graph to represent the relationship between users and items can incorporate novelty into the recommendation algorithm [5, 8]. However, [5] lacks discussion on hyperparameters tuning and other novelty score metrics, while [8] does not explore different graph structures or edge weighting schemes that can improve the recommendation algorithm.

Because we are working with a high-dimensional graph (many feature dimensions), a mutual proximity graph can help solve the problem of hubness that occurs

*All authors contributed equally to this research.

in high-dimension spaces [11]. However, this approach was only tested on a dataset with a limited number of music genres, and we want to be able to work with all available music genres.

Automatic visualization generation is one possible approach we could take for graph visualization [7]. It utilizes knowledge graph embeddings and features of the dataset to generate the most intuitive visualization. Additionally, the tool avoids the steep learning curve that many visualization libraries incur.

4 METHODS

4.1 Intuition

As seen in our previous survey of state-of-the-art methods, nearly none of them focus on the visualization aspect from the user's perspective. We believe that having a larger focus on visualization will allow us to not only concentrate more on the user's experience, but also allow for easier refining in the future by way of user feedback.

None of the methods that focused on visualization used force-directed graphs. One advantage with force-directed graphs is that they visually represent relationships in a way that is intuitive and easy to understand. They show both the overall structure of music recommendations and the individual connections. The nodes are also positioned to minimize the overall energy of the graph, meaning that strongly connected nodes are closer together. This helps users easily identify patterns and relationships between songs. They can also be easily customized to highlight different aspects that are more important to users because the size and color of nodes can be used to represent different attributes (ex. different colors for genre, increased/decreased size for similarity score).

Existing state-of-the-art methods utilize hypergraphs to perform music recommendations. An existing state-of-the-art publication [6] constructs a hypergraph with song features as hyperedges (ex. users, artists, releases, and tags). In our implementation, we collected additional song features such as danceability, energy, key, loudness, acousticness, instrumentalness, tempo, duration, time signature, and more.

We decided to use random walks as our baseline traversal algorithm for recommendations as it simpler than the machine learning methods seen in previous literature and is computationally less expensive than both machine learning methods and feature-based traversal. A random walk on a graph is an algorithm that begins at some vertex on the graph, and at each time step, moves to another vertex. It can be used to pick nodes that are close to a source node based on some probability distribution. We do not anticipate the random walk algorithm to perform more accurately than state-of-the-art methods, however it provides us with a baseline and allows for future experimentation with other algorithms, whether it be a graph traversal algorithm or a machine learning model.

4.2 Approaches

4.2.1 Data Collection. We looked into two main sources for our data: (1) Million Song Dataset and (2) Spotify. The Million Song Dataset included a complementary downloadable dataset called Last.fm which included over 500k tracks with song-level tags and similarity. The song and artist names in the Last.fm dataset were mapped to a Spotify song id and an artist id. These identification numbers would be used to to query Spotify API endpoints to collect additional song features which included danceability, energy, key, loudness, acousticness, instrumentalness, tempo, duration, time signature, and more.

To clean the data, we performed the standard techniques of dropping empty or zeroed out columns. Missing values would be replaced with a 0 or -1 depending if 0 was considered a valid value for the particular column.

In this aspect of our approach, our main innovation is performing data integration and merged the Last.fm dataset with the Spotify data. The combined data was use to generate our hypergraph.

4.2.2 Algorithms. A hypergraph is a generalization of graphs where a single edge can connect to multiple nodes.

We created the hypergraph by using two types of nodes: (1) song node, (2) artist node.

Every song node has attributes like name, id, artist, tags, dancability, energy, and more. Each node that corresponds to a source node has at least one edge that points from itself to a target node. These edges are referred to as "out edges" within the graph and are stored for each source song, while "in edges" are stored for each target song.

The inclusion of an artist node differentiates a regular graph from our hypergraph. We chose to use the artist feature for a hyperedge (due to issues with our other features in our dataset), but other song features could be used to create additional hyperedges in the hypergraph. From our sample dataset, if a certain threshold is is met by the number of songs released by an artist, a node for that artist will be created. These artist nodes will have both in and out edges for every song by them in the dataset. Essentially, the edges connecting to the hyperedge nodes are bidirectional.

We began by implementing a standard random walk on a standard graph to provide recommendations for a given song. In a random walk, you perform the following:

(1) Start the traversal from a source node, which in this case is the song chosen by the user. Initialize the similarity scores from the source node to every other node in the graph as 0 in a dictionary. Initialize an array storing the path the random walk took.
(2) Get the neighbors of the current node.
(3) Calculate the probabilities of choosing each node by using the similarity index between songs as weights. The probability of choosing a neighbor node from the current node is equal to similarity index between the current node and neighbor node divided by the sum of the similarity indices the current node has with all of its neighbor nodes.
(4) Randomly choose a neighbor node to go to based on the transition probabilities calculated in the previous step. Add this node to the array that stores the path of the random walk.
(5) For each neighbor node, add the probability calculated for that node to the similarity score for that node in the dictionary.
(6) Repeat steps 2-5 for your defined number of steps.
(7) Sort the similarity scores in the dictionary in descending order. Return the array denoted the random walk path and the similarity scores from the dictionary.

However, because we chose to work with a hypergraph, the random walk algorithm needed to be modified to pass through the feature nodes of a hypergraph (ex. artist nodes).

Because our goal was to get song recommendations and not artist recommendations, we had to check within the random walk algorithm that the node's attribute was "song" and not "artist." If the attribute was "artist," we still traversed the edges but we would run an additional iteration of the algorithm since the previous iteration yielded a node that wasn't a true song.

In this aspect of our approach, our main innovation is performing random walks on a hypergraph. Although this idea existed already, we could not find publications that utilized this for music recommendation.

4.2.3 Visualization Tool Design Choices. A visualization tool for a graph-based music recommendation system is not seen in this field of research. We hope our approach in creating a visualization tool can allow the user to easily understand how our system generates music recommendations.

Our initial step involved creating a force-directed graph to visualize recommendations. The data for the graph includes seven columns: (1) source, (2) source artist, (3) target, (4) target artist, (5) recommendation value, (6) source tags, and (7) target tags. The data is created with the top-K recommendations for a user, which are generated via the random-walk algorithm described earlier. The nodes represent recommended songs, and the edges are between songs that are considered similar according to the Spotify data set. The recommendation value is the corresponding value under the 'similars' column in the Spotify data set that denotes which songs are similar and how similar they are. The edges are darker if the songs are more 'similar' and lighter if they are less 'similar'. The tags are the genres of the song and the value for each tag is how much that song corresponds to that genre. These tags are displayed in the tool tip when the node is hovered over.

When creating the force-directed graph, the forces applied to the nodes are determined by:

- link: a force that draws the nodes towards their linked nodes
- center: a force that draws the nodes towards the center of the SVG container
- x: a force that draws the nodes towards a specified x-coordinate (in this case, the x-coordinate of the center of the SVG container)
- y: a force that draws the nodes towards a specified y-coordinate (in this case, the y-coordinate of the center of the SVG container)

- charge: a force that applies a repulsive force between the nodes to prevent them from overlapping

The links between nodes are drawn with a grey stroke if the value is 0 (they are similar), and drawn with a dashed stroke if the value is 1 (they are not similar).

Each node is represented with a circle and text (the song name). The radius of the circle is larger if there are more links attatched to it. The color of the circle depends on the number of links that the node has, where the colors of the nodes are on a linear color scale.

Our tools UI consists of three main sections:

- User input and interactive HUD
- Graph display
- Node information display

The User input and interactive HUD section allows the user to run our recommendation algorithm. This will show the traversal of the data visually while returning a list of the top-K recommendations for the user.

The Graph display feature uses d3's force simulation feature to produce a graph using songs as the nodes with edges based on the chosen graph construction method. The interactive features mentioned above are reflected on this graph.

Finally, the Node information display section offers more in-depth information about a selected song. This includes the name, artist, and if it is recommended.

The graph also allows for users to drag a node around the graph while other nodes move with it.

5 EXPERIMENTATION & EVALUATION

5.1 Questions

- Are the visualizations intuitive and easy to understand?
- Is the graph visually appealing?
- Do users like new recommendation system (aka do they like the recommended songs)?
- Given 50% of a playlist, how accurately can it recommend songs to match the other 50% of the playlist? Note: this metric will be evaluated with mean average precision.
- How does the recommendation system scale with more users/songs?

5.2 Experiments and Observations

Note the first three questions are subjective and, as such, require user feedback, while the last question will be investigated objectively.

Our team surveyed 20 students across campus about our music recommendation visualization tool on three metrics:

- Intuitiveness: Is it easy to understand and use our tool?
- Visual Aesthetics: Does our tool catch your eye? Is it visually appealing?
- Recommendations: Do you like the songs that it recommended?

We discovered that most users found the tool to be easy to use. However, the most popular feedback we received was if it was a user's first time on the website, we should include a "tour" of the site, teaching them how to effectively navigate and use our product.

As for visual aesthetics, most users complimented our color palette and liked how there was a gradient for edges that showed how similar songs are. They also liked how they could hover the nodes to see the different genres.

Our visualization tool's recommendation capability was limited due to difficulties encountered with data collection (*addressed in the Discussion section).

For the last question, asking if the subjects liked the songs that it recommended, subjects said they enjoyed most of the recommendations, but they didn't actually know most of the songs because the data set is so old. Another problem is some subjects had trouble picking songs to start the recommendation from because they didn't know any of the songs in the data set.

To measure the ability of our music recommendation system, we decided to test it by providing it 50% of a playlist and asking our system to generate the missing 50%. The result is evaluated with mean average precision.

Consider the two main features that go into recommendation accuracy: hypergraph construction and the traversal algorithm. Our experiments will consist of trials of the given task with varying construction methods or algorithms. The task will be to predict the remaining half of a user's or thematic playlist's songs given the initial half. We will use mean average precision (MAP) and recall as metrics to measure success. Given previous results in similar experiments and the volatility of

user playlists, we expect approximately 0.20 MAP or recall to be a reasonable upper bound.

In our experiments, given a playlist with 20 songs, we take 50% of the playlist and attempt to predict the other half. After running the experiment for 100 iterations, we achieved an average top-5 MAP of 0.02841 and an average top-10 MAP of 0.04840. In our experiments, top-k means that we take the top-k predictions and if any of the recommendations match, we count it as a successful recommendation. Although we achieved results in our experiments, these are not comparable to [6] because in their hypergraph implementation, they utilized embeddings.

6 CONCLUSION

We successfully created a hypergraph-based music recommendation system that utilizes a random walk to recommend songs to a user.

We also successfully made a UI with a force-directed graph to better visualize the recommendations.

According to surveyed students, our recommendation system was easy to visualize and understand, and the recommendations matched their music taste. However, our main suggestion was to include a tour of the site to explain how to navigate the site.

7 DISCUSSION

Our biggest constraint was time. The dataset we chose to work with was extremely large and it would take several days to collect all the song feature data with the Spotify API (assuming no issues with API rate limits). Additionally, we faced limitations with Spotify API's rate limits and missing values from Spotify's database. Due to these issues, we were only able to collect enough data to create a small sample subset for our project and demonstration. The limitations, implications, and future work are described below.

For developers using Spotify API for personal projects, Spotify limits the number of calls made to its API within a set time frame. Any additional calls are met with a "Retry-After X seconds" error. As a result, when attempting to gather data, we could only collect song features for around 2000 songs (out of 500k+ songs) approximately every 5 hours.

Another issue we faced was that Spotify's database didn't contain approximately 50% of the songs in the Last.fm dataset, meaning we couldn't find the song id or an artist id for a particular track from Spotify. This significantly reduced the number of potential entries in our final dataset. On top of that, Spotify's database didn't contain song feature data for every track, further decreasing the size of our dataset.

Due to the challenges we faced with our dataset, our experiments and evaluation results will not necessarily reflect how our recommendation algorithm will work in reality.

Our visualization tool displays a complete subgraph when it shows the user how the random walk retrieved the top-K recommendations. An improvement that we can make to reduce the number of song nodes shown, is to only show random walk path and the nodes directly adjacent to the random walk. Therefore, the visual will be less cluttered.

An important feature of any recommendation system is its scalability. In most use cases, recommendation systems are utilized in situations with large amounts of data. In the best case scenario, we expect our recommendation algorithm to run with a similar time complexity and memory (for user/song similarity score) to increase at a quadratic rate ($O(n^2)$). In addition, the algorithm and visualization tool will definitely be scalable. However, if we take the dataset issues into account, our music recommender system may not be as scalable due to the lack of song feature data. A potential future extension is to utilize machine learning models to extract song features (ex. genre prediction, audio analysis for acousticness/instrumentalness, etc.) or manually extract features (ex. tempo, key signature, etc.).

As far as potential implications and future plans go, we hope that our visualization will ultimately serve as a stepping stone for creating a better consumer-friendly tool for song recommendations. We believe that seeing an intuitive visualization with several informative (but not too detailed) recommendations at once based on a wide selection of user-selected criteria can help consumers explore their musical interests far more than just directly recommending a single song or compiling a playlist of recommended songs. In the future, assuming data can be successfully collected for song features and the visualization tool is improved to display only the necessary nodes, our visualization tool could be connected to mobile devices and be modified to sync directly with each user's profile on apps such as Spotify, Apple Music, and Pandora. This way, consumers could easily switch from their music streaming app to the

recommendation app and avoid the hassle of having to switch use another device/webpage.

8 WORK DISTRIBUTION

Our finalized plan of activities included four phases:

(1) Data Collection & Processing
(2) Hypergraph Construction & Recommendation Algorithm Implementation
(3) Visualization Tool Implementation
(4) Results & Analysis

While working towards our project progress report, we realized that phases (1) - (3) could be completed in parallel better utilize each team member's time. Therefore, we allocated 4 weeks of time for phases (1) - (3), and 2 weeks of time for phase (4).

Each team members work distribution are as follows:

- Aniruddha → random walk implementation; results and analysis
- Austin → Last.fm and Spotify data collection and integration; results and analysis
- Devan → visualization tool hosting setup and implementation
- Heidi → visualization tool implementation and experimentation
- Pranav → data cleaning; hypergraph creation; random walk implementation; results and analysis

All team members contributed approximately an equal amount of work.

REFERENCES

[1] Sander Dieleman Aaron van den Oord and Benjamin Schrauwen. 2013. Deep content-based music recommendation. *2013 Neural Information Processing Systems (NIPS)* (2013).

[2] Jiajun Bu, Shulong Tan, Chun Chen, Can Wang, Hao Wu, Lijun Zhang, and Xiaofei He. 2010. Music Recommendation by Unified Hypergraph: Combining Social Media Information and Music Content. In *Proceedings of the 18th ACM International Conference on Multimedia* (Firenze, Italy) *(MM '10)*. Association for Computing Machinery, New York, NY, USA, 391–400. https://doi.org/10.1145/1873951.1874005

[3] Janneth Chicaiza and Priscila Valdiviezo-Diaz. 2021. A Comprehensive Survey of Knowledge Graph-Based Recommender Systems: Technologies, Development, and Contributions. *Information* 12, 6 (2021). https://doi.org/10.3390/info12060232

[4] Tiffany Zhan Raymond Zhang Calais Fossier Robyn Markarian Carter Chiu Justin Zhan Laxmi Gewali Paul Oh Ferdos Fessahaye, Luis Perez. 2019. T-RECSYS: A Novel Music Recommendation System Using Deep Learning. *2019 IEEE International Conference on Consumer Electronics (ICCE)* (2019).

[5] Ruixing Guo, Chuang Zhang, Ming Wu, and Yutong Gao. 2016. A Graph-Based Novelty Research on the Music Recommendation. *2016 International Conference on Computational Science and Computational Intelligence (CSCI)* (2016), 1345–1350.

[6] Valerio La Gatta, Vincenzo Moscato, Mirko Pennone, Marco Postiglione, and Giancarlo Sperlí. 2022. Music Recommendation via Hypergraph Embedding. *IEEE Transactions on Neural Networks and Learning Systems* (2022), 1–13. https://doi.org/10.1109/TNNLS.2022.3146968

[7] Haotian Li, Yong Wang, Songheng Zhang, and Huamin Qu. 2022. A Knowledge Graph-Based Approach for Visualization Recommendation. *IEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS* 28 (2022), 195–205. https://doi.org/10.1109/TVCG.2021.3114863

[8] Kuang Mao, Gang Chen, Yuxing Hu, and Luming Zhang. 2016. Music recommendation using graph based quality model. *Signal Processing* 120 (2016), 806–813. https://doi.org/10.1016/j.sigpro.2015.03.026

[9] Dipanwita Paul and Subhra Samir Kundu. 2019. A Survey of Music Recommendation Systems with a Proposed Music Recommendation System. *Advances in Intelligent Systems and Computing* (2019).

[10] Sebastian Raschka. 2016. MusicMood: Predicting the mood of music from song lyrics using machine learning. (2016).

[11] Keigo Sakurai, Ren Togo, Takahiro Ogawa, and Miki Haseyama. 2022. [Paper] Deep Reinforcement Learning-based Music Recommendation with Knowledge Graph Using Acoustic Features. *ITE Transactions on Media Technology and Applications* 10, 1 (2022), 8–17. https://doi.org/10.3169/mta.10.8

[12] Tingting Zhang and Shengnan Liu. 2022. Hybrid music recommendation algorithm based on music gene and improved knowledge graph. *Security and Communication Networks* 2022 (2022), 1–11. https://doi.org/10.1155/2022/5889724

[13] Yuhang Zhang, Jun Wang, and Jie Luo. 2020. Knowledge Graph Embedding Based Collaborative Filtering. *IEEE Access* 8 (2020), 134553–134562. https://doi.org/10.1109/ACCESS.2020.3011105