A Decade of Song Covers Knowledge Graph

Duyen Nguyen, Naicih Liou

1. Introduction

Listening to music has been one of humankind's most popular leisure activities. Over the years, many songs have been created, recorded, and published to serve this purpose. People have also found joy in remixing and covering songs from other original songs. One song can have many different versions in different genres, languages, or singers. Despite its popularity, it is still challenging to find cover songs of an original song or original songs and covers of an artist. There is a lack of search engines available to query song covers. To solve this problem, we built a search engine with the power of the knowledge graph called A Decade of Song Covers Knowledge Graph (ADSC).

ADSC extracts data from three data sources: SecondhandSongs (www.secondhandsongs.com) for song information, Last.FM (www.last.fm) for singer information and Wikidata (www.wikidata.org) for songs with awards won or nominated. Due to computational expenses as well as time constraints, ADSC only focuses on songs within the period of time from 2012 to 2022.

ADSC not only provides query service for artists, and songs but also helps users to discover other versions of songs. We believe this search engine enables users to explore songs and covers.

2. Technical Challenges

a. Data Acquisition

Scrapy was our first choice for data crawling, which did a great job of scraping singer information from Secondhandsongs and Last.FM. However, when using Scrapy for crawling all song information from Secondhandsongs, we encountered the dynamic website problem. To be specific, Secondhandsongs changes the output displayed based on the search input instead of transferring to another page. To combat this problem, Selenium was used to mimic user interactions.

Moreover, our initial plan was to scrape TicketMaster for future events performed by singers. Unfortunately, we encountered the web-blocking problem when using Selenium. That made us change the data source from TicketMaster to Wikidata, in which we wrote a SPARQL query for songs with awards won and/or nominated within the past 10 years.

b. Entity Linking (SecondhandSongs - Wikidata)

Entity linking was employed in matching data from SecondhandSongs and Wikidata. The matching rules include matching by the name of the song and singer, by the string token of singers, and by the first name of singers. Blocking by initials of the singer's name was used before entity linking for more efficient matching.

i. Data Inconsistency

The first problem we encountered was data inconsistency. In the beginning, there were only 17 entities matched under Exact Matching of the name of the song and singer. After looking into the problem, we found out that some singers cooperated with other singers. This leads to the data inconsistency problem as the singer in wikidata was presented in one name but the name of the singer in SecondhandSongs was presented as singer and co-singers.Here is the example of songs with singer and co-singers:

- “My Shot” - Lin-Manuel Miranda, Anthony Ramos, Daveed Diggs, Okieriete Onaodowan, Leslie Odom Jr.
- “Payphone” - Maroon5 featuring Wiz Khalifa
- “Stay” - Rihanna feat. Mikky Ekko

This problem was solved with regular expressions. Patterns of featuring singers were found and applied to the data preprocessing. The singers of each song are divided into **singer** and **co-singer**.

ii. Entity Linking Rules Selection

Another problem is finding a good matching rule for entity linking. Matching by Jaro–Winkler distance and Levenshtein distance was applied in this case. The final matching rule is to match the Jaro–Winkler distance of the song name and a threshold of 0.8 was used. We used **F1 score** as validation, which reaches **0.8275**. Table 1 shows the experiment of finding the best matching rule. The reason Jaro–Winkler distance of song name was chosen as it has the same performance as others is that the rule is simple, easy to understand, and explainable.

Matching Rule	F1 Score
score_singer_jaro	0.769
score_song_jaro	**0.8275**
0.8*score_song_jaro + 0.1*score_singer_jaro + 0.1*score_singer_tokens_levenshtein	**0.8275**
0.6*score_song_jaro + 0.2*score_singer_jaro + 0.2*score_singer_tokens_levenshtein	0.7692

Table 1. Entity linking rules and the relative F1 scores

c. Ontology and KG construction

When we first created the ontology, both original songs and cover songs were in the same class of song and had the relation of 'originally_sung_by'/'was_covered_by' with the singer. Moreover, we needed to make sure the release dates of the original song and cover song were correctly linked to the singer and the song (Figure 1). However, RDFLib doesn't support specifying the property of the relationship, which makes attaching the release date to 'sung_by' relation not feasible. Even though Neo4j can specify the property of the relationship, using the initial ontology map wouldn't be feasible. We then decided to improve our ontology by differentiating songs into two classes, original songs and cover songs (Figure 2). The release date will be added as entity value to either the original song or the cover song.

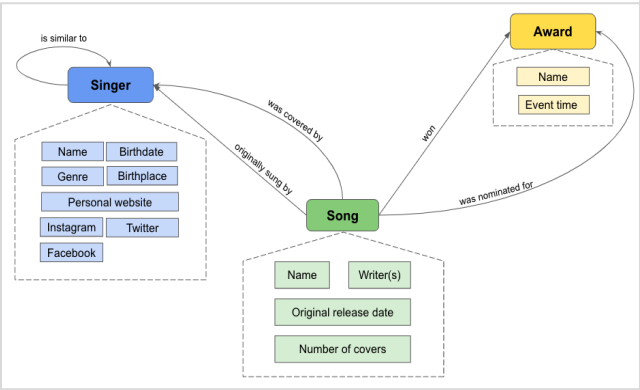

Figure 1. Initial ontology design

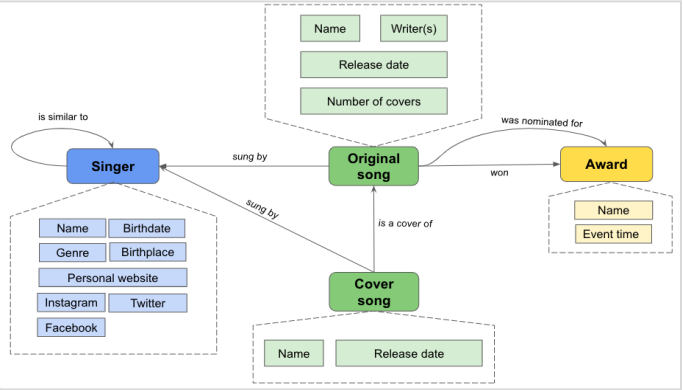

Figure 2: Finalized ontology design

Moreover, the same song title could be totally different songs, therefore, when creating URI for each entity, we had to make sure we assign a unique URI to each song so that songs with the same title wouldn't get mixed up with each other.

After using the finalized ontology design as a map for creating triples in ttl format using RDFLib, we loaded ttl file into Neo4j for further queries. The relations and nodes in Neo4j were successfully imported as well as correctly displayed. Cypher query was then used for testing whether the triples were accurately created, accurate results were yielded when querying, indicating our ontology and KG construction were up to expectation.

3. Lessons Learned

Building a search engine using the power of a knowledge graph is challenging but also very rewarding. We were able to apply KG techniques into practice and solve real-world problems. Specifically, using entity linking to match entities from data sources, creating triples using RDFLib, using NeoSemantics to load ttl file into Neo4j for Cypher query and graph visualization, and building query service through Flask and Neo4j.

We also learned how to construct an efficient workflow for building knowledge graphs, which could be beneficial for future projects. In this specific project, the workflow is as the following, Scrapy, Selenium, and SPARQL query were utilized to extract data from 3 data sources mentioned in the introduction. Then entity linking and blocking were employed with Python and RLTK library to match entities from different sources. Data were preprocessed to create ontology with RDFLib. Finally, a web application was connected with the knowledge graph through Flask and Neo4j to provide query service with Cypher. Through the use of all these tools, we learned the management of data, handling graph-based data entities, querying knowledge graphs, building an interactive web application, and creating visualizations for these entities.

This project also allows us to learn how to establish a mechanism to help users to explore the information about cover songs, and the relationships between covers and the original song. Moreover, users can use our application to follow singers via social media links as well as learn more about songs with awards won or nominated. Even though our search engine works properly as intended, we noticed that our knowledge graph is biased towards songs and singers in the language English, which we believe resulted from the bias from original sources due to the representation of English-speaking contributors. We hope to further collect more data from foreign sources to enrich our knowledge graph and to make it more complete and representative.

Demo Video: https://youtu.be/lU1cMm5DUjg
GitHub Repo: https://github.com/NaicihLiou/A-Decade-of-Song-Covers-Knowledge-Graph