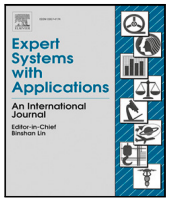
Expert Systems With Applications 229 (2023) 120347

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



I am all EARS: Using open data and knowledge graph embeddings for music recommendations

Niels Bertram [a], Jürgen Dunkel [a], Ramón Hermoso [b,*]

[a] Hannover University of Applied Sciences and Arts, Germany
[b] University of Zaragoza, Spain

ARTICLE INFO

Keywords:
Music recommender
Knowledge graphs
Graph embeddings
Recommender systems
Explainability

ABSTRACT

Music streaming platforms offer music listeners an overwhelming choice of music. Therefore, users of streaming platforms need the support of music recommendation systems to find music that suits their personal taste. Currently, a new class of recommender systems based on knowledge graph embeddings promises to improve the quality of recommendations, in particular to provide diverse and novel recommendations. This paper investigates how knowledge graph embeddings can improve music recommendations. First, it is shown how a collaborative knowledge graph can be derived from open music data sources. Based on this knowledge graph, the music recommender system *EARS (knowledge graph Embedding-based Artist Recommender System)* is presented in detail, with particular emphasis on recommendation diversity and explainability. Finally, a comprehensive evaluation with real-world data is conducted, comparing of different embeddings and investigating the influence of different types of knowledge.

1. Introduction

In the last decade, streaming music over the web has become the dominant way of consuming music (Friedlander, 2015). Music streaming platforms give users access to millions of songs from millions of artists. But finding content that matches one's interests and preferences is becoming a major concern. Especially in the music domain, users want to discover new bands and music styles they have not heard before to broaden their horizons. Exploring huge music libraries without any guidance or pre-filtering is clearly overwhelming.

Music recommender systems aim at recommending items in the music domain such as songs or music artists. Following, we sketch a couple of typical approaches used to implement recommender systems that, in turn, will be relevant for the understanding of the paper:

- *Collaborative filtering* is the best known and most-established recommender technology in commercial systems (Aggarwal, 2016), and has been widely used for music recommendation. The basic idea of collaborative filtering is that users who have liked similar artists and songs in the past are likely to share a preference for the same music. Following this assumption, collaborative filtering-based approaches recommend music items that similar users have liked. Users are considered similar if they have a similar listening history, which can be indicated by matching ratings or by the items consumed in the past. However, collaborative filtering suffers from the cold start problem (Schedl et al., 2018): there is not enough information about new users or music items in the system to derive meaningful recommendations for them. Moreover, it tends to popularity bias, recommending more popular items that are popular with everyone, and disregarding new or little-known items.
- *Knowledge graph-based recommender systems* make use of a graphical representation of the domain knowledge (Chicaiza & Valdiviezo-Diaz, 2021). A knowledge graph defines semantic relationships between entities relevant to the domain and the objects to be recommended. In the music domain, knowledge graphs could consider music artists, songs, albums, music labels, genres and more. In addition, known relations between users and music can be included in the graph, representing pieces of music they have interacted with in the past. The goal of knowledge graph-based recommendation systems is to find matches between users and items based on the given knowledge.
  Recent advances in the field of *graph embeddings* have led to the increasing popularity of knowledge graph embedding-based recommender systems (Wang et al., 2017). Graph embeddings transform the nodes of a graph into a latent vector representation so that the structural information of the graph is preserved. Working with graph embeddings instead of the graph

* Corresponding author.
E-mail addresses: mail@nielsbertram.de (N. Bertram), juergen.dunkel@hs-hannover.de (J. Dunkel), rhermoso@unizar.es (R. Hermoso).

https://doi.org/10.1016/j.eswa.2023.120347
Received 19 January 2023; Received in revised form 22 March 2023; Accepted 30 April 2023
Available online 12 May 2023
0957-4174/© 2023 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

itself is usually more efficient and allows application of vector-based mathematics such as calculating distances between vectors. Knowledge graph embedding-based recommendation systems make use of graph embeddings of the underlying knowledge graph to recommend items to users. They can overcome the traditional challenges in recommendations, namely the cold start problem and the challenge of providing meaningful explanations for recommendations.

In this paper, we investigate how music recommendations can benefit from a knowledge graph embedding-based approach. We propose the music recommender system *EARS (knowledge graph Embedding-based Artist Recommender System)*, which is based on knowledge graph embeddings and uses open data from the music domain. In particular, our main contributions are:

(A) *Knowledge graph:* Using open data sources, we develop a knowledge graph that contains the appropriate information for a music recommender to enable high-quality recommendations. We claim that good recommendations require two types of information: (i) semantically rich data about the music domain providing sufficient knowledge about music artists and bands, (ii) historical data on users' listening behavior, reflecting users' attitudes and feelings towards artists. This approach leads to a collaborative knowledge graph that combines content-based knowledge about artists with collaborative knowledge that connects users with similar preferences. The generation of the knowledge graph involves the choice of appropriate open data sources, feature engineering in the selection of relevant data, and the integration of different datasets. Furthermore, data sampling is required to reduce the dataset size for further processing. To the best of our knowledge, there is no comparable publicly available knowledge graph for the music domain, that contains, in particular, semantically rich data about artists and data about user behavior.

(B) *Embeddings:* Which embedding method works best for a given knowledge graph depends largely on its specific properties. Therefore, different embedding methods will be evaluated and compared to identify which are best suited for the EARS system. Using a standard library,[1] various embeddings are computed for our specific music knowledge graph. In particular, we determine the recommendation quality and runtime behavior on the EARS knowledge graph. Furthermore, different hyper parameters are compared and selected for the most promising methods.

(C) *Recommender:* We present the recommendation system EARS, which takes advantage of the graph embeddings of the underlying knowledge graph. The EARS recommendation mechanism consists of the following main building blocks: (i) user profiles described by users' listening histories, (ii) similarity of two artists computed by using their corresponding embeddings, (iii) similarity of candidate artists to a given user history. Unlike other work, EARS explicitly addresses the diversity of recommendations by using a bounded-greed selection approach. Moreover, we show how recommendations can be explained by exploiting the semantically rich knowledge contained in the knowledge graph.

A comprehensive evaluation of our approach is conducted with real-world data to gain insights into the performance of the recommendation system. Using various performance metrics, we evaluate the recommendation quality of EARS. In particular, we study the impact of the embedding methods as well as different recommendation parameters. Furthermore, we investigate the extent to which different domain-specific knowledge affects the recommendation results. For this purpose, we compare an EARS variant that uses the entire knowledge graph with all art-specific data with a variant that uses only the collaborative user data.

Overall, EARS offers a unique and flexible mechanism tailored to the recommendation of music artists that, in contrast to other work, is based on open data and explicitly addresses recommendation diversity and explainability.

The paper is organized as follows. In Section 2 we give a brief overview of music recommender systems and their major challenges. Section 3 discusses shortly the basic concepts of knowledge graphs and graph embeddings. Then, Section 4 presents EARS in detail and provides a detailed description of all included components. The results of extensive experiments evaluating our approach are presented in Section 5. Subsequently, related work is presented in Section 6. Finally, we summarize the paper and discuss some future lines of work in Section 7.

2. Music recommendation

This section provides a basic understanding of the music domain. First, an overview of how music is consumed today is given. Then, we discuss the basic data entities that are relevant in the music domain. Afterwards, we take a look at open data sources in the music domain that can provide knowledge graph-based recommender systems with the data they need. Finally, we discuss the most common use cases and challenges in music recommender systems.

2.1. Music consumption

Nowadays, music streaming services provide access to huge catalogues of music. As of 2022, Spotify offers a library of 80 million songs and claims to be the most popular audio streaming service with a total of 433 million user including 188 million subscribers.[2] Other popular music streaming platforms are *Tidal*, *Apple Music*, *Amazon Music*, *SoundCloud* and *Deezer*.

Back when access to music was still limited by ownership of a physical medium such as vinyl or CD, the music available was restricted by the access to such mediums or by the music selections of radio stations. Now that the majority of all recorded music is available at any time for anybody, the way music is heard and discovered has drastically changed. Not only do music listeners have a huge amount of music to choose from, but they also have to make the hard decision of what to listen to from the wide range of music on offer. Discovering music has also changed: before, record stores and radio hosts used to pre-select music for consumers to buy or listen to. Nowadays, music streaming services have to perform the task of filtering the available music to present users with the music they like to listen to. Given the almost complete availability of music nowadays, music consumption would hardly be possible without recommendation systems that pre-select the music relevant to a user from the existing offer.

2.2. Entities in the music domain

In order to develop a knowledge graph, it is first necessary to identify the fundamental entities in the music domain to establish a common vocabulary:

- A *recording* can be described as the result of producing music.
- A *track* (also called *song*) is a continuous, self-contained piece of music, which is always related to a single *recording*.
- A *release* is a one-time edition of a recording distributed to consumers on a specific date and by a specific label. It is stored on a medium (e.g. vinyl or CD) and can be of type album, compilation or single.

[1] https://karateclub.readthedocs.io.

[2] https://newsroom.spotify.com/company-info/.

- *Labels* (usually synonymous with record companies) are responsible for the production, manufacture, promotion and distribution of recordings.
- An *artist* can refer to an individual musician or a group of musicians as in a band. Furthermore, other music professionals such as producers or sound engineers are also referred to as artist in the music domain.

For instance, the band *Pink Floyd* is an *artist* which recorded the *recording Wish You Were Here* in 1975. The recording consists of five different *tracks*. *Brian Humphries* is the recording engineer of the recording and could thus viewed as another *artist* involved. The first *release* of the recording was published as an album by the *label Columbia* in 1975 on vinyl. As of 2022, the recording *Wish You Were Here* by *Pink Floyd* has a total of 93 *releases* by a variety of different labels on all types of available media.

Metadata

The data discussed so far relates to music in terms of entities involved in making, representing and distributing music. Metadata provides additional information, e.g. about an artists name, a tracks name, the year of recording, the length of the recording. Metadata is embedded in a music file such as an mp3-file representing a music track. The *de facto* standard for storing meta-data in music files is *ID3*. *ID3* was first released in 1996 and defines a standard for adding track name, artist name and album name to an mp3-file. In its current version ID3v2, it also supports longer metadata entries such as song lyrics (Morris, 2012). Generally, all kinds of information regarding a music item can be embedded in a music file using ID3 tags.

One type of information in the music domain that is often used and discussed is *genres*. Music genres classify music that share a distinctive musical language into common categories (Lena & Peterson, 2008). In addition to the music characteristics, additional factors such as nation or gender can be included in the classification (Brackett, 2016). Genres not only serve the industry in delivering new products to its target audience, it can also be useful for a consumer in terms of discovering new music.

2.3. Open music data sources

A key issue for a knowledge graph is how to populate it with data. This section presents the two open data sources we selected to build the music knowledge graph.

MusicBrainz

MusicBrainz[3] provides public information and metadata about music items. Its data is maintained and curated by a community of 252,000 active editors and includes about 2 million artists, 3.3 million releases and 220,000 music labels. Besides being a source of open music information, MusicBrainz provides unique MusicBrainz Identifiers (MBIDs) that can be added to identify releases or artists. The MBID can be used by other applications or datasets to link to a resource in the MusicBrainz database. Popular datasets such as the Last.fm dataset (discussed below) references entities such as music artists by using the MBID. MusicBrainz provides a complete data dump of its underlying PostgreSQL database, which includes tables for entities such as artists, releases and labels.

Last.fm data

Last.fm[4] is an excellent platform for collecting data about music listening behavior. It allows users to track the digital music they listen to locally on their devices or on services like Spotify or Tidal. The tracked information can be used to provide users with statistics about their listening behavior. Furthermore, users' listening histories are made publicly available, which makes them very useful for music recommender systems (Guo et al., 2020). One popular dataset based on Last.fm is the *Celma* dataset (Celma, 2010). The data includes music tracks from 359,000 users playing about 17.5 million tracks by 294,000 artists. Most artists have an MBID that points to the artist in MusicBrainz.

Other datasets

There are also other datasets that could be used for building a music knowledge graph. An obvious option would be *DBpedia*,[5] a data source for general knowledge that contains far fewer music-specific resources than MusicBrainz and is therefore less suitable. Another alternative would be *Discogs*,[6] which serves as a database and a marketplace of music releases. Although Discogs contains even a larger amount of data than MusicBrainz, it lacks more detailed attributes about artists, such as the region they come from, a founding date, and associated genres, all of which are included in MusicBrainz. Also, user behavior can more easily be linked to MusicBrainz, since Last.fm uses MBIDs to reference artists. Therefore, MusicBrainz was chosen instead of Discogs.

Besides open data sources, there are industry APIs offering data for the music domain, e.g. the Spotify API.[7] While the information provided by such APIs may well be of value due to the enormous scale at which services such as Spotify operate, access is usually very restricted.

2.4. Music recommender systems

Recommendation systems for music have to face different use cases and related challenges.

Use cases

In the most common use cases of music recommendation, either artists or songs are recommended. The latter can be recommended as a sequence of songs, known as playlist. For creating personalized playlists, music is selected according to the user's personal preferences (Schedl et al., 2018). A similar use case is the automatic playlist continuation, were songs are added to an already existing playlist, producing a potentially infinite stream of matching music (Chen, Yang et al., 2016).

When recommending artists, the main goal is to find an ordered list of artists that match a user's profile. A user profile should specify the user's interests, which can be defined by the user's listening history (Celma, 2010).

Challenges

Recommender systems have to face various challenges, some of which are particularly relevant in the music domain.

- The *cold start problem* may occurs when a new item such as an artist, song or user is added to a system (Schedl et al., 2018). Collaborative music recommendation systems are based on a user's listening history. Because such a history is missing for new users, or because new artists or songs do not yet appear in user histories, no recommendations can be made for them.

[3] https://musicbrainz.org/.

[4] https://www.last.fm/.

[5] https://www.dbpedia.org/.

[6] https://www.discogs.com/.

[7] https://developer.spotify.com/documentation/web-api/.

- *Diversity and Novelty*: especially in the music domain, users want to discover new bands and music styles they have not listened to before. Therefore, the diversity and novelty of the recommended items are a key issue in music recommender systems. Generally, novelty measures the likelihood that given recommendations have not been seen by a user before. Diversity considers how different the elements of a recommendation list are. A list of recommendations that matches a user's profile very closely, but in which all items are very similar to each other, may not be of much value to the user (Ziegler et al., 2005). In most cases, diversity is considered as dissimilarity between items in a recommendation list
- *Explainability* describes the ability of a recommender system to give explanations for why a recommendation was given to a user or why the system thinks the recommendation may be relevant to a user. Explanations can increase the acceptance of recommendations, improve trust as well as the user's loyalty towards the system and altogether lead to higher satisfaction (Celma, 2010).

It should be noted, that further challenges have been discussed (Schedl et al., 2018; Silveira et al., 2019) that are of less focus in this paper. *Temporality* addresses temporal aspects of recommender systems. Often, user preferences change over time, so a user's older history may be less interesting. *Serendipity* deals with the ability of a recommender system to make surprising recommendations to a user. *Popularity bias* describes the effect that more popular items are generally more likely to be recommended. The popularity bias is closely related to the long tail data distribution (Abdollahpouri et al., 2019; Kowald et al., 2020). Considering all items, there are a few very popular items and a huge set of more or less unknown items. Ignoring the long tail may lead to less innovation and diversity in music recommendations. The popularity bias could also lead to further biases and discrimination as recent research suggest: e.g. existing biases towards white and male artists may be reinforced by recommender systems prone to the popularity bias (Werner, 2020).

3. Knowledge graph-based recommendations

Knowledge graph-based recommender systems calculate recommendations on base of a graphical representation of the domain knowledge. For the music domain, a knowledge graph should contain the key entities such as artists, songs, labels and genres. To model user preferences, the graph can also include relationships between users and the music they have interacted with in the past. Since such user-element interactions are also used in collaborative filtering approaches, this is referred to as collaborative knowledge graphs (Wang et al., 2019).

3.1. Knowledge graph concepts

Knowledge graphs are data graphs used to convey knowledge about the real world, with nodes representing entities of interest and edges representing different relations between these entities (Hogan et al., 2022). They focus on knowledge instances rather than a schema that describes the data and its structure. Usually, knowledge graphs have a shallow schema with small degree of formalization (Ji et al., 2021; Paulheim, 2016).

Representing knowledge as a graph has several benefits: Complex and varying relations between entities can be naturally represented by the graph edges. In addition, knowledge graphs do not necessarily need a predefined schema, so data can evolve more flexibly. Furthermore, methods established in graph theory such as link prediction and finding shortest paths can be directly applied to knowledge graphs.

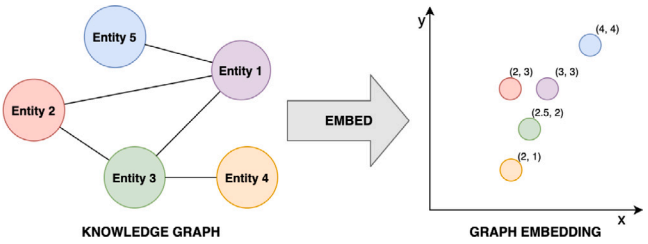


Fig. 1. Example of graph embedding into the 2-dimensional space.

Path-based approaches

A general approach to calculate recommendations is to find items that are similar to those that the user has liked in the past. The simplest definition for determining the semantic similarity between two entities in a knowledge graph is the length of the shortest path between these two entities (Guo et al., 2020; Jia et al., 2018). This method, sometimes simply denoted as the *path* method, is rather naive and has major shortcomings: some parts of a knowledge graph, e.g. about a particular music genre, may be more detailed, so that paths here are longer than elsewhere in the graph, leading to a bias in the similarity measures. Furthermore, the *path* method does not distinguish types of entities and relations, but some paths between two entities may be more meaningful than others in regards of semantic similarity. In a music knowledge graph, for example, artists linked by common countries are arguably less semantically similar than those linked by the same genre.

There are several approaches to deal with these shortcomings. The *wpath* method tries to weight paths based on the type of connected entities and other context information (Jia et al., 2018). Another popular approach is the use of manually defined *meta-paths* that represent a meaningful connection between two nodes in a knowledge graph (Sun et al., 2011). Meta-paths can also be weighted to give more weight to certain paths than others (Shi et al., 2015).

While path-based recommendation approaches feel rather natural and intuitive, it has been shown that they are difficult to apply in practice (Chicaiza & Valdiviezo-Diaz, 2021). In addition to the manual effort required to build and maintain the knowledge graph, metapath-based approaches require further guidance from domain experts to create and update metapaths.

Embedding-based approaches

The goal of knowledge graph embeddings (KGE) is to simplify the way a knowledge graph can be processed while maintaining its structure (Chicaiza & Valdiviezo-Diaz, 2021). Knowledge graph embeddings represent entities of a knowledge graph in terms of vectors in a n-dimensional space. An example of embedding a graph into the 2-dimensional space is shown in Fig. 1.

The general approach to generating recommendations using entity embeddings is to represent both users and items by latent vectors. The distances between the items already known to a user and items unknown to her can be used to create recommendations. The list of recommendations contains yet unknown items that are within a short distance from the user's previously popular items (Guo et al., 2020).

The main advantage of embedding-based recommendations is that they allow entities such as items or users to be represented as vectors, which are often easier and more efficient to process compared to complex graph structures. Distances in vector space can be calculated using established distance measures and the vectors could also be used as feature vectors for further machine learning tasks.

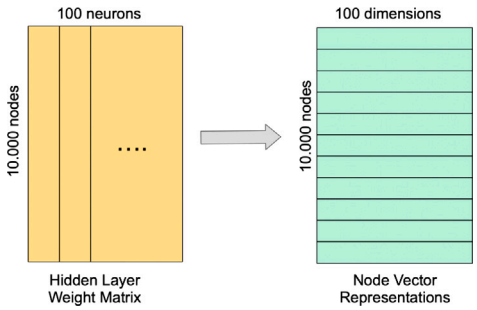


Fig. 2. Skip gram for representing node vectors in *DeepWalk*.

3.2. Knowledge graph embeddings

Graph embedding methods can be distinguished into random walk-based methods (also called *graph sampling-based*) and factorization-based methods. Below we will briefly discuss some of them.

3.2.1. Random walk-based methods

Random walk-based methods are based on exploring the neighborhood structure in a graph. They are well suited to capture such a structure, i.e. nodes close to each other in a graph get similar vector representations. Since they operate in local neighborhoods, they are especially useful in cases where a graph can only be partially observed or is too large to process in its entirety (Goyal & Ferrara, 2018).

DeepWalk

`DeepWalk` (Perozzi et al., 2014) is inspired heavily by the natural language processing approach `Word2vec` (Mikolov, Chen et al., 2013; Mikolov, Sutskever et al., 2013) and applies the same idea to graphs instead of text. As a basic principle, it uses random Walks to obtain sets of neighboring nodes in the graph. Random walks traverse a graph randomly from a given target node. During a walk, paths are created by uniformly choosing a neighbor of the last visited node as the next node on the path until a walk length n is reached. Random walks are repeated for each node m times, so that a graph of N nodes is transformed into $N \cdot m$ paths of length n containing the nodes and their neighbors. The node paths generated by the random walks are then used to train a neural network that can predict the probability of reaching a node v_i if certain neighboring nodes have been visited during the random walks. To achieve this, sequences of nodes contained in a moving window of size w in the node paths are considered as input to the neural network. The neural network follows the so-called *skip gram* model, a simple neural network with a single hidden layer. The trick of the *skip gram* model is that the neural network is not used for prediction, but that the learned weight matrix of the hidden layer is taken as the vector representation of the nodes. After training, the weights of the hidden layer are taken as latent vector representations of the graphs nodes. The number of neurons in the hidden layer determines the dimension of the resulting node vectors. Fig. 2 illustrates how the weight matrix of the hidden layer is used as the vector representation of node. For more on `DeepWalk` see Perozzi et al. (2014).

Node2Vec, Diff2Vec and Walklets

`Node2Vec` (Grover & Leskovec, 2016) is an extension of `DeepWalk` that introduces biased random walks, meaning that decisions about which node to visit next are no longer completely random. This bias is implemented by parameters which allow the random walk to alternate between breath-first and depth-first graph search, with the goal of preserving community structure and structural equivalence. This results in nodes being strongly connected with each other (community structure) as well as nodes having similar roles in communities (structural equivalency).

Another random walk-based approach is `Diff2Vec` (Rozemberczki & Sarkar, 2018), which addresses the problem that random walks propagate very slowly and revisit the same nodes multiple times, resulting in redundant information. The proposed solution of `Diff2Vec` is to create so-called diffusion graphs of the neighborhood of a target node v by iteratively adding random neighbor nodes to a graph initialized with node v. In those diffusion graphs, Euler tours are computed to obtain node sequences covering all nodes of the diffusion graph. This approach should result in better coverage of a node's neighborhood, allowing `Diff2Vec` to work with smaller neighborhood samples than other random walk-based methods.

`Walklets` (Perozzi et al., 2017) is a variant of `DeepWalk` in which multiple scales of relationships between nodes are explicitly taken into account. In `DeepWalk`, the proximity between two nodes not directly connected is only implicitly represented via paths through other nodes. Instead, `Walklets` introduce skipping steps in random walks to allow higher order proximity representation.

Besides the methods discussed so far, the `NetMF` method (Qiu et al., 2018) unifies `Node2Vec`, `DeepWalk` and two additional approaches (`LINE` and its extension `PTE`) into a matrix factorization framework.

3.2.2. Factorization-based methods

The general idea of factorization-based embedding methods is to represent a graph in a matrix and factorize the matrix to obtain embeddings. Factorization-based methods differ in the type of matrix in which a graph is represented and the method used to factorize the matrix.

An early factorization-based embedding method is `laplacian eigenmaps` (Belkin & Niyogi, 2001). It represents a graph by its laplacian matrix, which is calculated by subtracting the adjacency matrix of a graph from its degree matrix. Then, the laplacian matrix is factorized using eigen-decomposition. Generally, this approach tries to minimize the distance between node embeddings of nodes which are close to each other in the graph.

`Geometric Laplacian Eigenmap Embedding` (`GLEE`) is a novel method based on the original idea of `laplacian eigenmaps`. But unlike the latter, which aims to minimize the distances of embeddings for close nodes in the original graph, `GLEE` tries to find embeddings by leveraging the so-called simplex geometry of the laplacian (Torres et al., 2020).

`GraRep` is a method that combines the idea of random walks with matrix factorization (Cao et al., 2015). The model captures the different k-step relations between graph nodes for different values of k, to create a global transition matrix. The resulting matrix is factorized using singular value decomposition (SVD).

Instead of utilizing eigen-decomposition as in `Laplacian Eigenmaps` or singular value decomposition as in `GraRep`, the `NMFADMM` method (Sun & Fevotte, 2014) makes use of non-negative matrix factorization. `BoostNE` (Li et al., 2019) is a factorization-based method that allows multiple embeddings to be learned and combined into a final embedding, using an approach motivated by the *gradient boosting* learning algorithm. `RandNE` (Zhang et al., 2018) applies Gaussian random projection in the factorization which has been proven to be more computationally efficient than other methods and enables operation on very large graphs.

Besides random walk and factorization-based graph embeddings, there are several other types of embedding methods. `NodeSketch` (Yang et al., 2019) is a method that employs a data-independent hashing technique called *sketching* to generate embeddings. Also worth mentioning are `TransE` (Bordes et al., 2013), as well as methods using graph convolutional networks (`GCN`) to learn graph embeddings (Goyal & Ferrara, 2018). `GCNs` are applied in `R-GCN` (Schlichtkrull et al., 2018), and in `ConvE` (Dettmers et al., 2018).

Please note that a detailed presentation of all these approaches is beyond the scope of this paper, so we refer the reader to the original papers.

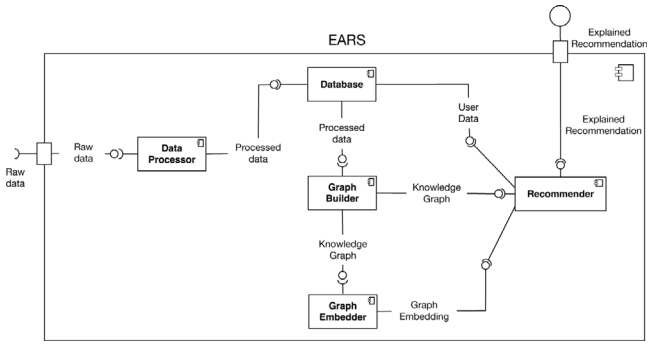


Fig. 3. Component diagram of EARS.

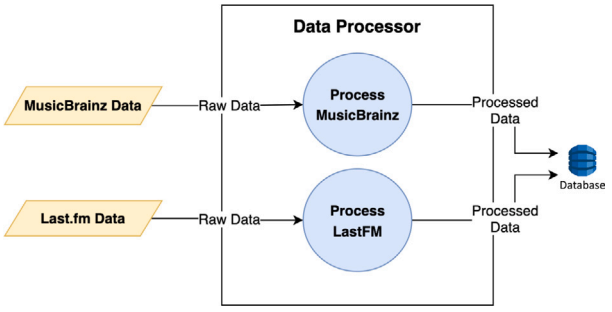


Fig. 4. The `Data Processor` component.

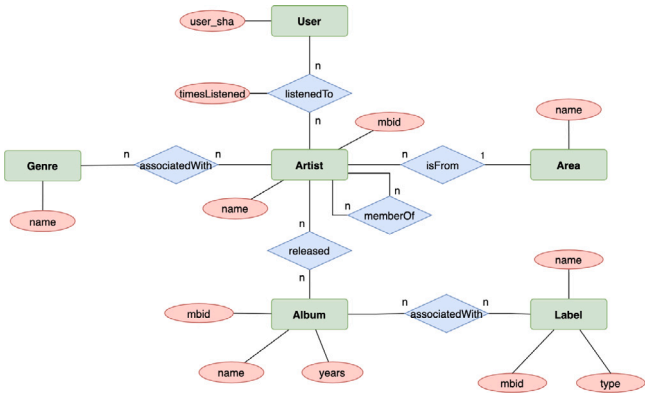


Fig. 5. Schema of the local database.

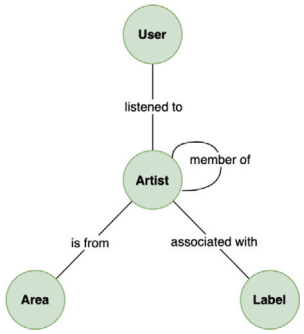


Fig. 6. Structure of the knowledge graph.

4. Knowledge graph Embedding-based Artist Recommender System (EARS)

The general design of EARS approach is depicted in Fig. 3. Following, different components are presented along with their corresponding functioning.

4.1. Data processor

The `Data Processor` component takes raw data as its input, extracts and processes domain relevant data and stores the latter in a local database for subsequent use by other components. As already mentioned in Section 2.3, *MusicBrainz* was chosen as the main data source for music meta-data and its relations, as well as *Last.fm* providing user listening behavior. The general data flow of the component can be seen in Fig. 4.

MusicBrainz data was dumped from the MusicBrainz database, including all main entities and their relations. The *Last.fm* dataset provides information regarding the listening history of thousand of Last.fm users. This data contains users, artist IDs and the number of plays each user has played a given artist. The artist IDs are MusicBrainz IDs, which makes it easy to merge the two datasets.

Fig. 5 presents the database schema resulting from the `Data Processor` workflow. This includes artists, genres, areas, albums, labels, users and their attributes and relations.[8]

Since the dataset contains about two million artists and over a hundred thousand users, labels, and areas, a sample of users is drawn to reduce the data size and make the subsequent steps, especially the knowledge graph constructing and the embedding calculation, feasible. The data size can be reduced by excluding those users who rarely used Last.fm and whose data is therefore less meaningful than that of more frequent users. Users relevant to the dataset can be defined by a minimum number of different songs or different artists they must have listened to. Moreover, the total sample size can be specified, defining a certain percentage of the original data to be randomly sampled.

4.2. Graph builder

To build the knowledge graph, a MusicData object is constructed from the data in the `Data Processor` database, which is then used to create a graph object by using the *networkx* library.[9]

Fig. 6 shows the structure of the knowledge graph. Nodes are represented by 2-tuples with a unique id and a dictionary of their attributes. Node attributes include their type (e.g. user, artist) and the related names (e.g. user name, artist name). For edges, the ids of the connected nodes are stored along with attributes that indicate the type of the represented relationship. The *networkx* library allows saving the graph in a variety of formats, including *graphML*.

As described in Section 4.1, the knowledge graph contains only a subset of all users in the Last.fm dataset. Accordingly, artists and related entities are only included in the knowledge graph if there is at least one user who has listened to that artist. The resulting knowledge graph can be viewed as a subgraph of the overall knowledge graph, including only those entities that are connected to the contained users.

Interestingly, the resulting graph forms some meaningful clusters that differ by genre and area: Artists tend to be located near the genre to which they belong. The same applies to areas. As an example, Finland, which is known for its metal scene, is located next to the genre *metal* and right in the center of metal bands.

[8] Note that Last.fm entries with MBIDs that do not exist in the MusicBrainz database are not considered. This might occur because the MusicBrainz data is up-to-date while the Last.fm dataset was created in 2010. Some MBIDs may have changed, merged or removed, resulting in obsolete entries in the Last.fm dataset.

[9] https://networkx.org/.

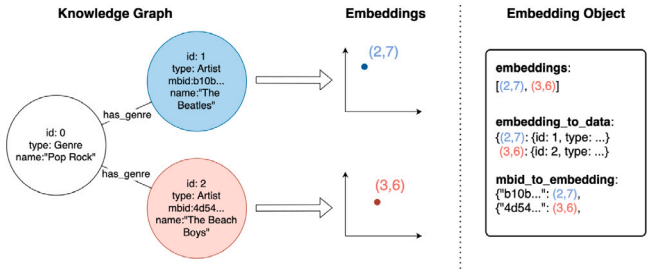


Fig. 7. Embedding example and its resulting embedding object.

The resulting graph is unweighted and undirected. All relations are unambiguous in its semantic meaning with respect to direction. Note that the undirected *memberOf* relation can only describe that two artists are related. To describe who is an individual artist and who is a band would require a directed relationship, which is probably not worth the additional complexity that a directed graph would introduce. The number of times an artist has been played by a certain user is integrated by another mechanism, as discussed later in Section 4.4.

4.3. Graph embedder

The purpose of the `Graph Embedder` component is to take a graph and compute an embedding which represents the nodes of the graph as vectors in a n-dimensional space. The results are saved in an Embedding object that stores a list of the embedding vectors and implements mappings from node embeddings to their node attributes and, vice versa, from artist MBIDs to node embeddings. Because an artist recommender requires only artists and their proximity to each other, the `Graph Embedder` component discards all non-artist nodes embeddings.

After experimenting with different graph embedding libraries, *karateclub*[10] was chosen for its many available embedding methods and its tight integration with *networkx*. An example of an embedding process is shown in Fig. 7. For illustration purposes, the artist nodes in the knowledge graph are mapped to a 2-dimensional space.[11] Each point in the vector space represents an embedding of an artist node.

4.4. Recommender

The main task of the `Recommender` component is to calculate recommendations for a certain user based on the embedding computed by the `Graph Embedder`. Furthermore, the `Recommender` should be able to provide explanations for a given recommendation by finding paths in the underlying knowledge graph. Recommendation process and the related data flow is shown in Fig. 8.

4.4.1. User profiles

A key issue of any recommender system is modeling the user profiles from the given user data. For music recommendations, a user's preferences can be characterized by the artists in her listening history, respectively their corresponding embeddings.

Fig. 9 shows a user profile represented by the corresponding graph embeddings. A certain user (depicted as blue circle) has a listening history containing different artists (depicted as red circles). The example shows a typical Last.fm user who is often interested in very different music genres. In this example, the set of the artists she has already heard can be categorized as jazz, folk, rock and classical music.

[10] https://karateclub.readthedocs.io.

[11] In the implementation, of course, much higher dimensional vectors are used.

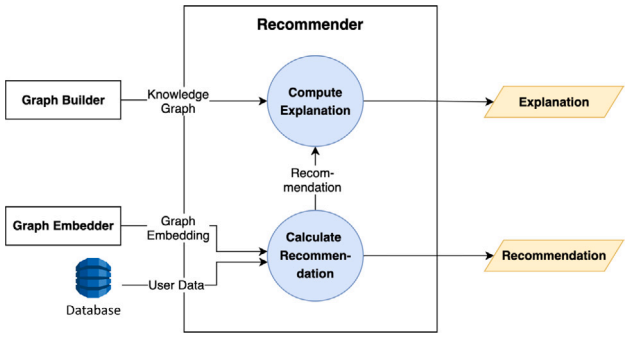


Fig. 8. The `Recommender` component.

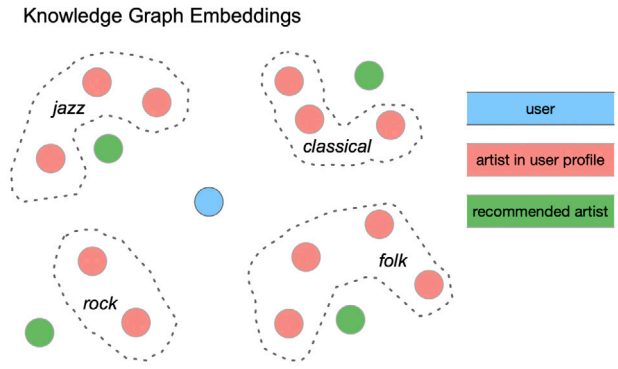


Fig. 9. Example: user profile and recommendations.

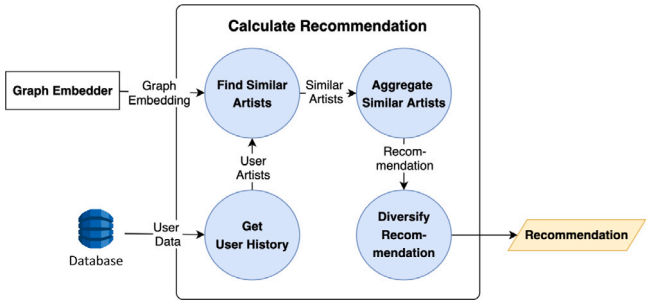


Fig. 10. The four steps of the recommendation process.

It is obvious that a suitable music recommender should take this diversity of interests into account. To enhance end-user satisfaction, a variety of artists should be recommended, representing all genres of music previously listened to. In Fig. 9 the green circles show some recommended artists that fit well to the respective clusters. This approach is in line with *user profile partitioning* presented in Zhang and Hurley (2009), where recommendations are based on user profile clusters. EARS follows the same idea as we will discuss in more detail in the next subsection.

4.4.2. Recommendations

Given a user u for whom a recommendation should be made, the recommendation process consists of four main steps, as shown in Fig. 10: (i) getting a user's listening history from the database; (ii) finding similar artists based on the produced graph embedding; (iii) aggregating the similar artists to a single list of recommendations and; finally (iv) applying diversification to that recommendation.

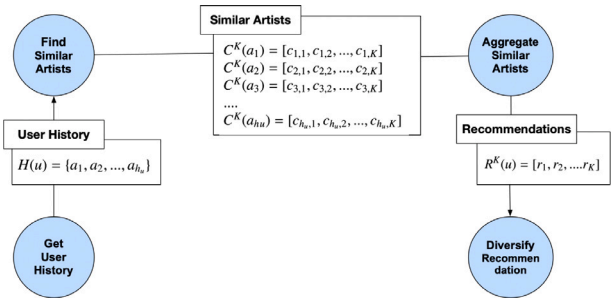


Fig. 11. Example of data in the recommendation process.

Artist	Distance
John Lennon	0.0
The Beatles	10.755525915983442
George Harrison	11.095350416524859
The Rolling Stones	11.113060925318106
Paul McCartney	11.845007380574549
The Who	12.269100541530843
Bob Dylan	12.595374814401485
Jimi Hendrix	12.697791517853751
David Bowie	12.72701405838986
Queen	12.764438833379142
Oasis	12.810402487657658

Fig. 12. The 10 nearest artists to *John Lennon* in the k-d tree.

(I) get user history. The user history $H(u)$ of user u is the set of all artists that she has listened to in the past,

$$H(u) = \{a_1, a_2, \ldots, a_{h_u}\} \tag{1}$$

where $a_i \in A$ is an artist from the set of all artists A known to the system, and h_u is the number of artists that u has heard so far.

(Ii) find similar artists. The goal of this step is to find for each artist $a \in H(u)$ from the user history $H(u)$ the list $C^K(a)$ of the K most similar artists to artist a.

$$C^K(a) = [c_1, c_2, \ldots, c_K] \tag{2}$$

where the order is descending by the distances $dist(c_i, a)$ between the artists c_i and a. The function $dist()$ corresponds to a distance metric in the vector space of graph embeddings, such as the Euclidean distance. Each c_i from $C^K(a)$ can be considered as a recommendation candidate matching an already known artist a.

The underlying idea is that artists that are close to each other in the knowledge graph are similar because they have relationships to the same or similar entities (e.g. label, genre, area, user). As the graph embedding preserves the structural information of the knowledge graph, artists close to each other in the graph also have embedding vectors close to each other in the vector space. The structure of the data involved in the recommendation process is illustrated by Fig. 11.

For querying nearest neighbors efficiently in a higher-dimension space, a k-d tree is used (Bentley, 1975), applying the idea of binary search trees to k-dimensional spaces. A k-d tree splits a k-dimensional space at each node, meaning that points to the left of a splitting hyperplane are located in the left subtree of that node and points on the other side of the hyperplane to the right. Utilization of the tree properties allows to find nearest neighbors in $O(log(N))$ where N is the total number of nodes in the tree. The k-d tree is initialized once with the embedding vectors of artist nodes when the `Recommender` component is initialized. For the nearest neighbor search, we use Euclidean distance as a metric for computing similarities.[12]

The number K of retrieved nearest neighbors of an artist already known by the user is one of the `Recommender` parameters and has influence on the recommendation results. Fig. 12 shows an example of querying the k-d tree showing the $K = 10$ most similar artists to *John Lennon*.

(Iii) aggregate similar artists. In the previous step, for each artist $a \in H(u)$ in the listening history of user u, a separate list of similar artists $C^K(a)$ is created by a k-d tree query. These separate lists of artists must now be combined into a single list of ordered recommendations.

To assess the extent to which a candidate artist c is suitable for recommendation, its distances to the artists in the user history are calculated. The calculated distances must be scaled to avoid being skewed by artists in denser areas of the vector space, which naturally should have smaller distance values to each other. Considering the set $C^K(a)$ of the K most similar artists to an artist a from the user history, we applied min–max scaling, resulting in a score value within the range $[0, 1]$, as shown in Eq. (3).

$$score(c, a) = 1 - \frac{dist(c, a) - min(C^K(a))}{max(C^K(a)) - min(C^K(a))} \tag{3}$$

For the function $dist()$ we have chosen the Euclidean distance, as mentioned before. The functions $min()$ and $max()$ return the minimum and maximum distance values, respectively, in the set of similar artists $C^K(a)$. Finally, we subtract this scaled value from 1 to give the most similar artist a high score of 1 and the furthest a score of 0.

It seems reasonable that artists who are similar to the favorite artists of a user will receive higher scores. To evaluate the popularity of an already heard artist a for user u, the popularity score p_score is calculated, which multiplies the score with a popularity factor $p_u(a)$ as shown in Eq. (4).

$$p\_score(c, a, u) = score(c, a) \times p_u(a)^w \tag{4}$$

where $p_u(a)$ is the scaled number of times user u has played artist a. The $p_u(a)$ values are also scaled with respect to the number of plays in the listening history of user u by using min–max scaling to obtain values in the range $[0.1, 1]$.[13] To control the impact of $p_u(a)$ on the score, a weight $w \in [0, 1]$, which serves as an exponent, is used. Since $p_u(a) \in [0.1, 1]$, smaller values of w generally have a smaller effect on the p_score because for small w, $p_u(a)^w$ approaches 1.

Artists can appear in multiple lists of similar artists $C(a)$, i.e., they are similar to more than one artist a in the user's listening history. For example, a user u may have listened to *The Beatles* and *Bob Dylan* in the past. For both, *The Beatles* and *Bob Dylan* the list of similar artists are retrieved and their scores are calculated. The results can show that *Paul Simon* is a similar artist to both *The Beatles* and *Bob Dylan*.

To calculate the total score that a candidate artist c has for a given user u, its p_scores for all artists from the user history are just summed up, as shown in Eq. (5).

$$total\_score(c, u) = \sum_{a \in H(u)} p\_score(c, a, u) \tag{5}$$

In terms of the given example, $p\_score(PaulSimon, BobDylan, u)$ and $p\_score(PaulSimon, TheBeatles, u)$ would be added to get $total\_score(PaulSimon, u)$.

To produce the final recommendation list $R(u)$ for user u, the $total\_score$ of each candidate artist c included in the set C^K is calculated, where $C^K = \bigcup_{a \in H(u)} C^K(a)$ is the set of all artists that appear in at least one $C^K(a)$ of any artist from the user history. Finally, the ordered recommendation list $R^K(u)$ for user u contains K recommended artists

[12] Of course also other metrics such as cosine similarity can be applied.

[13] with a value of 1 for the most played artist. Note that we used a lower bound of 0.1 instead of 0.0 in the scaling to avoid null scores due to small number of plays.

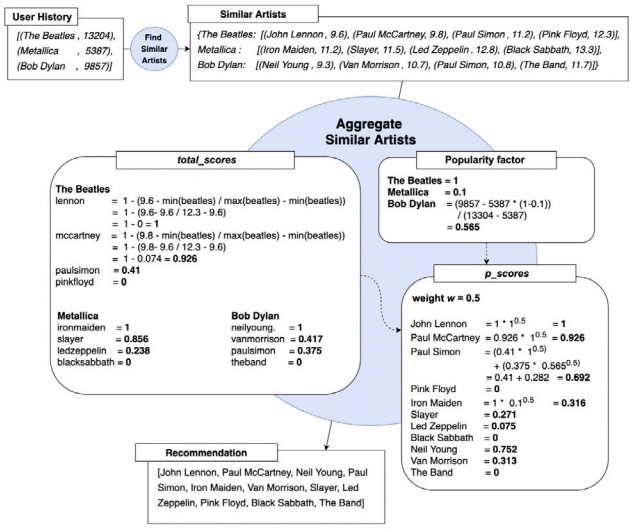


Fig. 13. Calculation of an example recommendation.

in descending order according to their $total\_score$ values, as shown by Eq. (6).

$$R^K(u) = [r_1, r_2, \ldots r_K] \tag{6}$$

A complete example of the recommendation process is shown in Fig. 13. Here, the considered user u has listened to the artists *The Beatles*, *Metallica* and *Bob Dylan*, playing each of them a certain number of times. First, for each of these three artists, the list $C^K(a)$ of the most similar artists with their corresponding distances is obtained by querying the k-d tree. For example, similar to *The Beatles* are *John Lennon* with a distance of 9.6, *Paul McCartney* with a distance of 9.8 and so on. Next, the number of plays of each artist a out of $H(u)$ are scaled to calculate the popularity factor $p_u(a)$. Then, for each artist c of $C^K(a)$, the $p\_scores(c, a, u)$ are computed. In the final step, $p\_scores(c, a, u)$ are aggregated to the $total\_scores(c, u)$. Since *Paul Simon* is a similar artist to both *The Beatles* and *Bob Dylan*, the two resulting p_scores are added to the $total\_score(Paul Simon, u)$. This example illustrates the process in a very small scale of just $h_u = 3$ artists in the user's history with $K = 4$ of similar artists to retrieve.

(Iv) diversify recommendation. The final and optional step is the diversification of the recommendations, as illustrated in Fig. 10. Diversification is applied by applying the `Bounded Greedy Selection` (BGS) algorithm, which makes use of intra-list similarities to increase diversity without sacrificing accuracy (Smyth & McClave, 2001). BGS combines the idea of a `Bounded Random Selection` and `Greedy Selection`.

Algorithm 1 Bounded Greedy Selection

$u \leftarrow$ considered user
$K \leftarrow$ length of the recommendation list
$b \leftarrow$ bound of artists to consider
$R = R^{b*K} \leftarrow b * K$ recommended artists accord. to Eq. (6)
$D \leftarrow R[0]$, move best recommendation to result set D
for i in $\{1, 2, ..., K\}$ **do**
  for each $r \in R$ calculate $diversity\_score(r, R, u)$
  sort R by $diversity\_score(r, R, u)$
  $D \leftarrow D + R[0]$
  $R \leftarrow R - R[0]$
end for
return D

Alg. 1 shows how BGS works. The algorithm is initialized with the considered user u, the length of the recommendation list K and a bound b. Then, a list R of $b * K$ recommendable artists according to Eq. (6) is used as a candidate set for the recommendation list. That is, it takes into account a multiple of the K recommendations that are finally given to the user u.

The basic idea of BGS is, to select out of R those K artists that have the best $diversity\_score$. These form the result set D with K artists that provide a tradeoff between similarity to the user's listening history and diversity. Using Eq. (7) the $diversity\_score$ for each artist r in R is calculated by combining the $total\_score$ from Eq. (5) and the $diversity$ metric as defined by Eq. (8).

$$diversity\_score(r, D, u) = total\_score(r, u) * diversity(r, D) \tag{7}$$

The $diversity(r, D)$ of an artist r is the average inverse scores to all other artists d_i in the result set D, as calculated in Eq. (8). This means that, on average, a highly diverse artist is as different as possible from all other artists already included in the result set D.

$$diversity(r, D) = \frac{\sum_{d_i \in D}(1 - score(r, d_i))}{|D|} \tag{8}$$

Finally, the set R is sorted according to the $diversity\_score$ and the artists with the highest score ($R[0]$) is moved from R to the result set D.

4.4.3. Explanations

The knowledge graph used in the recommendation system can be used to provide explanations for recommendations. As can be seen in Fig. 8, the explanations are computed from the knowledge graph G and the recommendations from the graph embedding of G.

The basic idea is that paths between an artist that a user u has already listened to and a newly recommended artist can serve as an explanation for why u might like this artist. For example, both artist nodes could be connected to the same label, genre or area.

A few decisions have been made to produce rather simple but still meaningful recommendations: edges between users and artists, representing which user listened to which artists, have been removed to keep explanations music-specific instead of "similar users also liked ..."-type of explanations.

Furthermore, only paths of length one are considered, i.e. instead of long (and potentially meaningless) paths, only common neighbors are looked at. Common neighbors result in explanations such as "*You may like X because you listened to Y, both of which are signed to label L and from country C*".

To keep it clear and simple, the maximum amount of recommendations for each artist can be set. Which explanations are selected can be determined by prioritizing certain relationships. For example, if the *memberOf* relation has first priority, it will always be displayed first if such a relationship exists. As a default, the priority is set to *member of*, *label*, *genre*, *area* in that order. The rationale behind that order is that more specific and distinguishing connections are prioritized since they probably convey a deeper meaning to a user.

To give an idea how EARS works, Fig. 14 shows a generated list with recommendations for a randomly chosen *Last.fm* user. At the top, artists which the user listened to in the past are listed. In the following recommended artists and explanations for why the artist may be of interest to the user are listed. Artists are ordered in decreasing order with the most relevant in first place. The actual length of the recommendation list is set to five in this example but could be longer in practice.

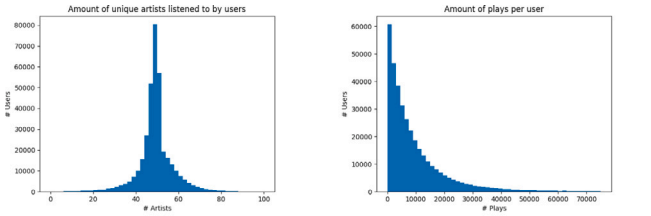
5. Experimental evaluation

In this section, we present an exhaustive set of experiments to test the suitability of the EARS approach.

To evaluate the quality of recommendations, we check if EARS is able to recommend artists that appear in the user's history. Here we proceed as follows: Each user u taken from the *Last.fm* dataset has a

```
The user listened to the following artists in the past:

Y & T                  Thin Lizzy             Twisted Sister         Led Zeppelin           Van Halen
Metallica              Red Hot Chili Peppers  Jamiroquai             Frank Zappa            Savatage
David Lee Roth         Deep Purple            Rush                   Sleepy Sleepers        UFO
Fatboy Slim            Queensrÿche            Krokus                 KISS                   Michael Schenker Group
Scorpions              Megadeth               Talking Heads          Primus

The following artists are recommended to the user:
==========================================

Black Sabbath
------------------------------------------
You may like Black Sabbath because you listened to Frank Zappa, both of which are associated with the label EMI
You may like Black Sabbath because you listened to David Lee Roth, both of which are associated with the genre rock
You may like Black Sabbath because you listened to Judas Priest, both of which are from United Kingdom
==========================================

AC/DC
------------------------------------------
You may like AC/DC because you listened to Frank Zappa, both of which are associated with the label EMI
You may like AC/DC because you listened to Twisted Sister, both of which are associated with the genre glam rock
==========================================

Rainbow
------------------------------------------
You may like Rainbow because you listened to Accept, both of which are associated with the label Hear No Evil Recordings Ltd
You may like Rainbow because you listened to Krokus, both of which are associated with the genre hard rock
You may like Rainbow because you listened to UFO, both of which are from United Kingdom
==========================================

Motörhead
------------------------------------------
You may like Motörhead because you listened to Queensrÿche, both of which are associated with the label EMI
You may like Motörhead because you listened to David Lee Roth, both of which are associated with the genre rock
You may like Motörhead because you listened to Iron Maiden, both of which are from United Kingdom
==========================================

Ozzy Osbourne
------------------------------------------
You may like Ozzy Osbourne because you listened to Jamiroquai, both of which are associated with the label Sony Music
You may like Ozzy Osbourne because you listened to Savatage, both of which are associated with the genre heavy metal
==========================================
```

Fig. 14. Explained recommendation for a random Last.fm user.



(a) Number of artists per user (b) Number of total plays per user

Fig. 15. Insights into the Last.fm data.

Table 1
Parameters for data sampling.

Data sampling parameters	
Sample size	0.05
Minimal number of plays	20.000
Minimal number of user artists	40
Test data split	0.5

Table 2
Statistics of the generated knowledge graph.

Total nodes	Artists	Labels	Areas	Genres	Users
26,135	19,953	2657	86	626	2351

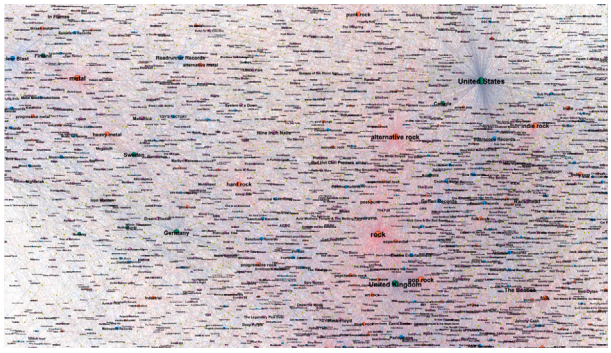


Fig. 16. Excerpt of the generated knowledge graph.

certain number of different artists in her listening history $H(u)$. For example, user u might have listened to 50 artists. This set of artists is randomly split by half between training data $H_{train}(u)$ and test data $H_{test}(u)$. EARS now uses the 25 artists from the training set $H_{train}(u)$ as user profile and creates from them the result set D, containing e.g. $K = 10$ recommended artists. Then it is checked how many of the artists from D also appear among the 25 artists of $H_{test}(u)$. If all 10 artists from D occurred in $H_{test}(u)$, this would imply a perfect precision of 1.0.

Different metrics will be considered to measure the performance of the approach. We will use some traditional metrics, such as *precision*, *recall*. Due to the nature of the scenario we use *precision@k* and *recall@k* only accounting for the first k artists resulting from the recommendation process. In the same line, we also calculate *false-positive-rate@k*, defined as the ratio between the number of false positive and the number of all negative cases of the first k items returned by the recommender, as well as *Mean Average Precision* (MAP).

We also compute the *Receiver Operating Characteristic* (ROC) curve and its associated *Area Under the Curve* (AUC), which represents the likelihood that a random relevant item is ranked higher than a random irrelevant one. In addition to these accuracy metrics, *diversity* and *novelty* metrics are calculated as well.

In order to obtain meaningful results, these are calculated by averaging the data obtained from several runs with different random samples, i.e. a number of n users is randomly selected m-times. An empirical evaluation of the recommender parameters is also carried out. After that, an exhaustive evaluation of the EARS recommender systems is provided, taking into account the different metrics aforementioned.

5.1. Knowledge graph setup

As discussed in previous sections, the creation of the knowledge graph is controlled by a set of parameters. To find suitable parameter values, the *Last.fm* data was analyzed. Fig. 15 gives insights on how the number of artists and the total number of plays are distributed for users in the *Last.fm* data. It shows: (i) how many users listened to a given number of artists (Fig. 15(a)) and; (ii) how many users have a given number of plays (Fig. 15(b)). It can be seen that the number of artists per user forms a normal distribution with a mean value around 50 artists. The number of plays per user forms an exponential distribution, showing that the majority of users recorded very few plays, while only few users played a lot of songs.

One goal of the parameter selection was to considerably reduce the size of the knowledge graph while preserving the significance of the data. Since users with few artists or plays are more likely to provide an incomplete picture regarding their listening behavior, it was decided to omit such users. The minimum number of artists per user was set to 40 and the minimum number of plays was set to 20,000. The sample size was set to 0.05 resulting in a data volume which allows building the knowledge graph and, in particular, computing the graph embeddings in a reasonable amount of time. The split of the test data was set to 50%, resulting in an equal amount of training and test data. An overview of all parameter values is listed in Table 1.

The sampled data generated by the `Data Processor` is used by the `Graph Builder` to build a knowledge graph and export the graph as a *graphML* file. The number of nodes by type is listed in Table 2.

Fig. 16 shows an excerpt of the generated knowledge graph with about 8000 nodes and 35,000 edges, which was created with the Gephi tool.[14]

5.2. Embedding methods evaluation

Since the graph embedding is a key aspect of the EARS recommender, it was decided to evaluate a series of different embedding methods to empirically choose the most suitable one. The embedding methods are first applied with the default parameters provided by the

[14] https://gephi.org.

Table 3
Parameters for evaluating embedding methods.

`Recommender` parameters	
Top-k per `User Artist`	100
Plays weight w	0
Evaluation parameters	
Number of runs m	5
Users per run n	100

Table 4
Results of embedding method evaluation.

Method	p@10	AUC	MAP	Time (s)
BoostNE	0.048	0.812	0.020	757
DeepWalk	**0.170**	0.862	**0.069**	302
Diff2Vec	–	–	–	>3600
GLEE	0.106	**0.892**	0.056	27
GraRep	–	–	–	>3600
LaplacianEigenmaps	0.086	0.867	0.045	734
NetMF	0.130	0.886	0.067	46
NMFADMM	0.001	0.737	0.003	47
Node2Vec	–	–	–	>3600
NodeSketch	0.033	0.785	0.014	217
RandNE	0.100	0.836	0.040	**4**
SocioDim	0.074	0.863	0.031	2013
Walklets	0.070	0.890	0.038	209

karateclub Python package, assuming that the defaults are reasonable chosen. Thus, results derived can give a good hint about which methods are promising and deserve further inspection. A time limit for the embedding computation was set to one hour. Since the size of the knowledge graph is already limited, it was decided that methods that take longer than this amount of time are likely to be rather inappropriate for larger graphs.

The value of K for Top-K selection was set to 100. The weight w for the popularity factor was set to 0, which results in $p_u(a)^w = 1$ always, completely eliminating the popularity factor for calculating recommendations. The effect of weight w on the recommendation results will be discussed later. As evaluation parameter, the number of runs was set to 5 and the users evaluated per run to 100. A summary of the parameter chosen for evaluating embedding methods can be seen in Table 3.

Table 4 and Fig. 17 summarize the embedding method evaluation. Table 4 shows values for *precision@10* (p@10), AUC, *Mean Average Precision* (MAP) and the execution times.[15] Fig. 17 shows plots of the ROC, a zoomed-in version of the ROC curve and the precision–recall curve. The zoomed-in ROC shows only the lower-left section of the whole curve, providing insights about how the curve behaves for small false-positive rates.

`Diff2Vec`, `GraRep` and `Node2Vec` took more than one hour and the embedding process was abandoned without further evaluation.

In terms of promising results, `DeepWalk` stands out by having the best values for precision@10 and MAP. Regarding AUC, other methods surpass `DeepWalk`. The zoomed-in ROC curve in Fig. 17(b) shows that `DeepWalk` outperforms other methods for lower false-positive rates. Similarly, up to a certain point, `DeepWalk` performs better in terms of the precision–recall curve.

Looking at the rest of the embedding methods, `NetMF` stands out with the second best evaluation results behind `DeepWalk`. Furthermore, `NetMF` needs considerably less time for embedding the graph compared to `DeepWalk`. As it was expected, `RandNE` is very fast compared to all other methods while still producing rather good results.

It was decided to run some further experiments regarding `NetMF` parameters to see if parameter tuning could improve the results considerably. This seemed especially interesting, because for both, `NetMF` and `DeepWalk`, the dimensionality of the resulting embedding vectors can be set. For `DeepWalk`, the default dimensionality was 128, while for `NetMF` it was 32, raising the question whether `NetMF` with higher dimensional embeddings could produce equal or better results than `DeepWalk`. Table 5 shows evaluation results for `NetMF` with 32, 64 and 128 dimensions. As expected, the time needed increases with more dimensions. Furthermore, the results did improve with 64 dimensions compared to 32 dimensions. However, `NetMF` could not reach the results of `DeepWalk` even with increased embedding dimensions.

Table 5
Evaluation of `NetFM` with different dimensions.

Dimensions	p@10	AUC	MAP	Time
NetMF 32	0.129	0.886	0.067	46
NetMF 64	0.141	0.888	0.070	76
NetMF 128	0.142	0.882	0.069	126

When evaluating the embedding methods, `DeepWalk` proved to be the most promising method in terms of finding a trade-off between the different metrics and execution time, and was therefore selected as the embedding method for further experiments.

5.2.1. *DeepWalk parameter evaluation*

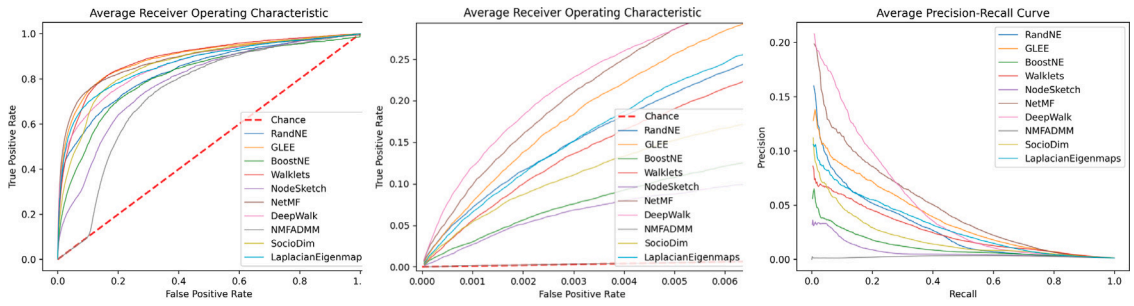
Experiments with the `DeepWalk` parameters showed that changes to most of the default *karateclub* parameters, such as the dimensionality of embeddings, did not significantly improve the results. Only longer walks gave better results, but also caused longer computation times. Therefore, different values for the walk number and the walk length are evaluated below. The remaining parameters of the experiments are similar to those used before and shown in Table 3 but with 200 instead of 100 users per run.

First, the walk length was varied with a fixed walk number of 10 walks per node. Embeddings were computed using `DeepWalk` for increasing walk lengths from 10 to 150 with a step size of 10. The results can be seen in Fig. 18.

Precision increases steeply up to a precision@10 of 0.15 at a walk length of 50 and then more gently up to 0.17 at a walk length of 100, after which there is no further improvement (see Fig. 18(a)). AUC increases relatively steadily from 0.840 to around 0.865 (see Fig. 18(b)). Results for `MAP` look similar to the precision@10 curve (see Fig. 18(c)). However, the actual values double from 0.035 at a walk length of 20 to 0.07 at around a length of 90. After that, MAP does not seem to increase significantly. At a walk length of 10, the MAP value is a little higher than expected, which could be caused by a statistical outlier. The increase in execution time is linear with increasing walk length. Doubling the walk length results in about twice the amount of time needed to compute the embedding (see Fig. 18(d)). Since both precision@10 and MAP seem to hit an upper limit at a walk length of about 100 and computation time should also be considered, a walk length of 100 was chosen for the following experiments.

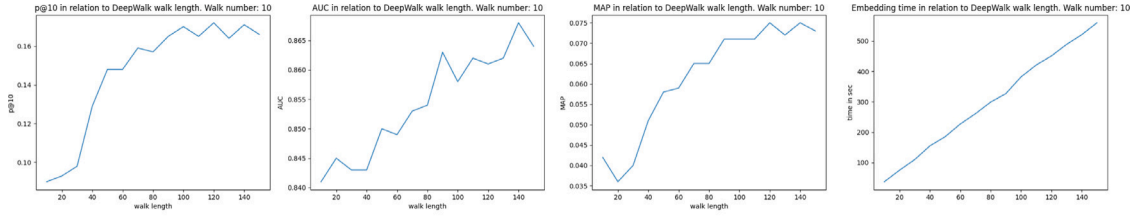
An analysis of the number of random walks for `DeepWalk` was also carried out. The results of the evaluation are presented in Fig. 19. As expected, all metrics raise when increasing the number of walks. AUC and MAP may hit a ceiling at around 80 walks, but further experiments with even more walks would be needed to be certain about that. precision@10 increases from about 0.162 at 10 walks to 0.172 at 100 walks. AUC starts at around 0.860 and ends at 0.880. Regarding MAP, values start at 0.072 at 10 walks and end up at 0.082 at 100 walks. Time increases linearly with the number of walks, again roughly doubling the time when doubling the number of walks. Since embedding time already went beyond one hour for 100 walks of length 100 it was decided to stop evaluation at 100 walks. Results for precision@10 indicate that values may have increased further for walk numbers beyond 100. Furthermore, evaluation results for AUC and MAP suggest that a ceiling was possibly hit at a walk length of 80. Based on the results of `DeepWalk` parameter evaluation, a `DeepWalk` embedding with 100 walks and a walk length of 100 was chosen as the graph embedding for all further experiments.

[15] Computed on a 2,7 GHz Intel Core i5, 8 GB RAM.



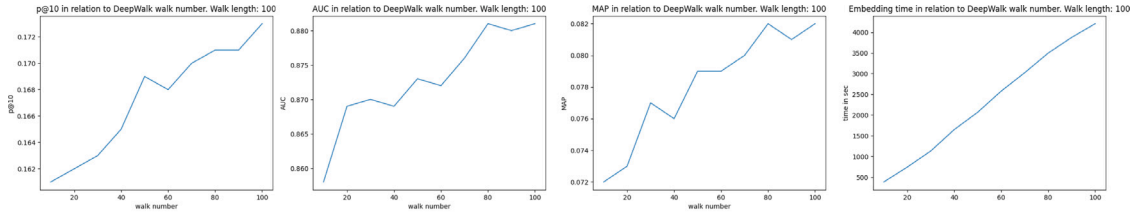
(a) ROC (b) Zoomed-in ROC (c) Precision-Recall Curve

Fig. 17. Evaluation results of embedding method evaluation.



(a) p@10 (b) AUC (c) MAP (d) Execution time

Fig. 18. Evaluation of DeepWalk walk length in relation to accuracy.



(a) p@10 (b) AUC (c) MAP (d) Execution time

Fig. 19. Evaluation of `DeepWalk` walk number in relation to accuracy.

5.3. Evaluation of top-K parameter

The K parameter for top-K selection was evaluated starting from 1 up to 200 with step size 5. The results of the top-K evaluation can be seen in Fig. 20. MAP increases steeply from around 0.03 with top-K at 1 to about 0.075 with K being set to 30. It reaches a ceiling slightly above with $K = 50$ with a MAP of 0.08. Thereafter, MAP stays relatively stable up to about $K = 160$, from where on performance in terms of MAP seems to be getting worse again. What can be said with confidence is, that MAP does not increase substantially after a K of 50. Regarding AUC, measurements also go up rather steep from 0.81 with $K = 1$ to 0.87 with $K = 60$. With higher values of K AUC seems to still increase, but not as significantly any longer. Querying more artists for the recommendation, which will most likely increase performance at higher values of K. precision@10 goes up rapidly to 0.165 with $K = 10$. Considering the discussed results, setting $K = 75$ seems a reasonable choice. MAP has hit its (assumed) ceiling by that point, AUC has done its initial climb and precision@10 is at around its best before values start to decline again.

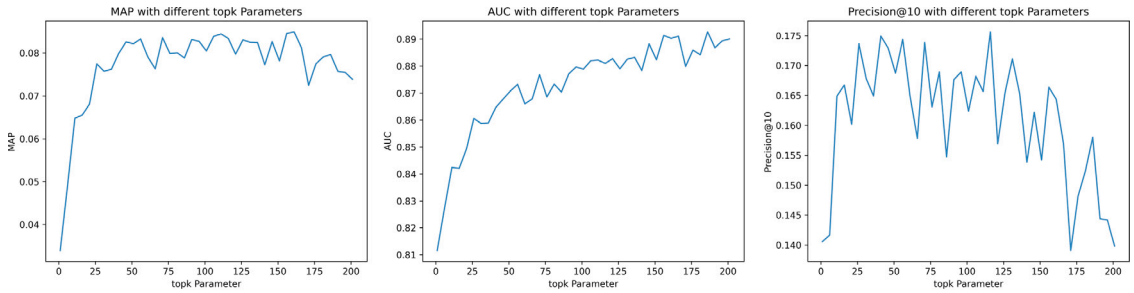
5.4. Evaluation of popularity weight parameter w

For the popularity weight w, evaluation started with a value of 0.0 and was incremented in steps of 0.05 up to 1.0, see Fig. 21. Regarding AUC, the value of w does not seem to have any real effect on MAP and precision@10. There are no clear trends and the two highest values at 0.6 and 0.7 could well be outliers due to the random selection of users. However, it looks like there may be a little upward trend towards the middle of the graphs followed by a slight drop off. Thus, choosing $w = 0.5$ may improve the recommendations a bit while most likely not leading to any worse recommendations. In general, it seems that in our approach the popularity of an artist for a user does not significantly affect the quality of recommendations. A possible explanation for this behavior could be that artists less often played by a user are just as important in representing the users preferences as the most played ones.

5.5. EARS evaluation

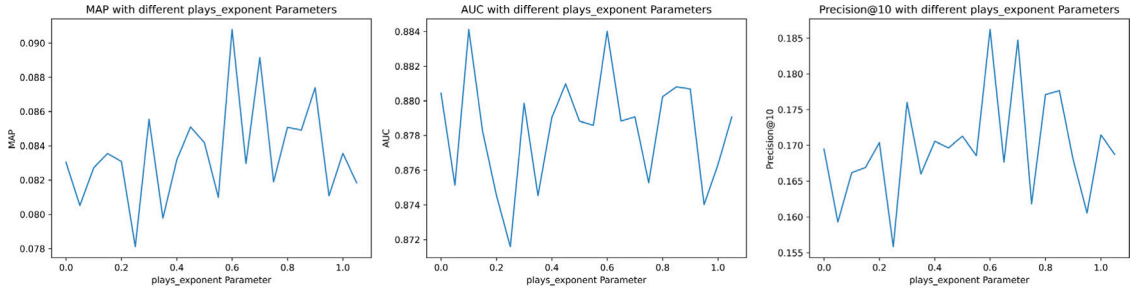
After trying to find optimal values for the different technical parameters of the recommender approach, the overall quality of the EARS system shall be investigated in more detail. Two types of recommendation approaches were examined, using the music knowledge graph fed by *MusicBrainz* and *Last.fm data*, and differing in whether the diversification step was performed (`mg-dwe-diverse`) or not(`mg-dwe`[16]). Diversification with *Bounded Greedy Selection* uses the parameter $b = 15$ while K corresponds to the length of the recommendation list. For example, to give a recommendation of $K = 10$ artists, the top $10 \times 15 = 150$ artists have been considered.

[16] Music graph deep walk embedding.



(a) MAP (b) AUC (c) p@10

Fig. 20. Top-k parameter evaluation.



(a) MAP (b) AUC (c) p@10

Fig. 21. Plays weight parameter evaluation.

5.5.1. Evaluating recommender variants

Since the connections between the users and the artists they listened to make up an important part of the knowledge graph, it could be possible that these relations are the main contributors to the recommendations. However, this would mean that EARS takes an approach similar to collaborative filtering, using a knowledge graph instead of a rating matrix. To gain insight into the extent to which music-specific domain knowledge contributes to the recommendations, a `lfm-dwe`[17] variant of EARS was additionally evaluated. In this case, EARS uses a different knowledge graph, which is solely based on *Last.fm* data, ignoring all music-specific knowledge from *MusicBrainz*. This means that the knowledge graph contains only users and artists that are connected according to their listening history, resulting in a bipartite user-artist graph. Apart from the knowledge graph, `lfm-dwe` works exactly like the recommender discussed in Section 4. Some preliminary experiments with the `lfm-dwe` variant showed that it works well with the same parameters as described in the previous sections.

Thus, for the final evaluation, the same parameters as shown in Table 6 were set for `mg-dwe`, `mg-dwe-diverse` and `lfm-dwe`. The parameter values used for or this set of experiments were derived from the results of the previous experiments (for the embedder parameters from Section 5.2, for the top-k parameter from Section 5.3 and for the w parameter from Section 5.4).

Furthermore, as baseline, a `rand-rec` approach was implemented, which recommends artists completely randomly. In other words, `rand-rec` has access to the entire pool of artists, but picks recommended artists randomly instead of doing any further calculations.

5.5.2. Evaluation results

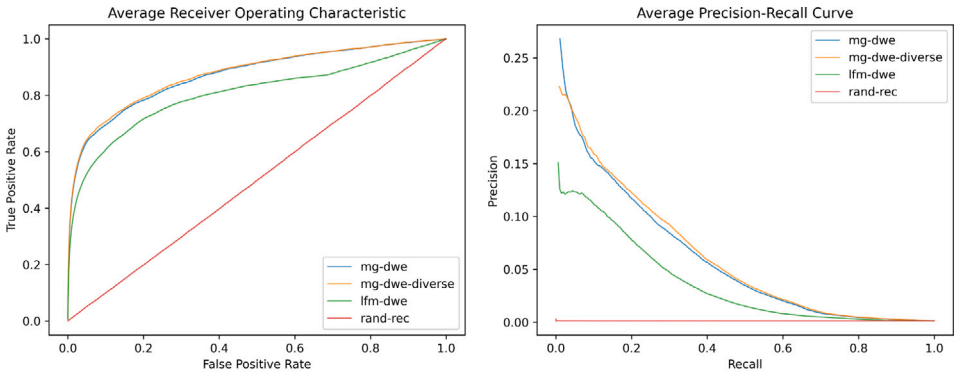
In this section, the evaluation results of the different variants of the EARS system are presented. To get a more accurate picture of the recommendation quality, we consider $precision@K$ and $recall@K$ for values of $K = 5, 10, 25$. Furthermore, $novelty$ and $variety$ were evaluated for $K = 10$ recommendations. The evaluation results are shown in Table 7 and Fig. 22.

Table 6
Parameter setup for EARS evaluation.

`Embedder` parameters	
Embedding method	DeepWalk
Walk length	100
Walk number	100
`Recommender` parameters	
Top-K per *User Artist*	75
Plays weight w	0.5
Evaluation parameters	
Number of runs m	5
Users per run n	200

Let us first take a look at the `mg-dwe` variant of EARS using both the *MusicBrainz* and the *Last.fm* datasets. $Precision@K$ is between 0.204 and 0.141 depending on K. Note that $p@10 = 0.176$ means, that on average, 1.76 out of ten given recommendations are relevant to the user. To understand the meaning of this value, let us look at how it is determined. The *Last.fm* dataset contains on average 50 artists in the listening history $H(u)$ of each user. This set of artists is split by half between training and test data. This means that if there are 50 artists in the listening history, 25 of them belong to the training data that the recommender knows, and the other 25 belong to the test data that the recommender does not know. The calculation of $precision@10$ checks how many of the 10 recommended artists refer to one of the 25 artists from the part of the listening history unknown to the recommender. Considering that a user's test data consists of only about 25 artists on average and there are about 20,000 artists in total, hitting almost 2 out of those 25 artists by selecting 10 artists out of 20,000 is a very good result. Recall values are between 0.040 at $K = 5$ and 0.116 at $K = 25$ for `mg-dwe`. Given that an AUC of 0.5 would represent completely random recommendations and 1.0 would be the perfect recommender which recommends only relevant artists (as long as they exist), a final AUC of 0.872 seems rather promising.

[17] Last.FM deep walk embedding.



(a) ROC (b) Precision-Recall Curve

Fig. 22. Evaluation results.

Table 7
Evaluation results.

	mg-dwe	mg-dwe-diverse	lfm-dwe	rand-rec
p@5	**0.204**	0.202	0.121	0.0014
p@10	0.176	**0.177**	0.123	0.0013
p@25	0.141	**0.142**	0.110	0.0012
r@5	**0.040**	**0.040**	0.024	0.000
r@10	0.069	**0.070**	0.049	0.001
r@25	**0.116**	0.096	0.108	0.001
AUC	0.872	**0.876**	0.805	0.499
MAP	0.083	**0.085**	0.057	0.002
novelty@10	1797	1791	2013	**2636**
diversity@10	0.253	0.289	0.311	**0.596**

Interestingly, when comparing `mg-dwe` and `mg-swe-diverse`, accuracy does not get worse with diversification. The evaluation results are very similar. As expected, `mg-dwe-diverse` increases diversity slightly over `mg-dwe` as measured by `diversity@10`. Note that `mg-dwe` already has a built-in mechanism for diversifying recommendations by making recommendations for each of the possible diverse artists in the listening history. Being able to diversify without actually sacrificing accuracy is an unexpected but welcome result which suggests that `mg-dwe-diverse` is preferable when compared to `mg-dwe`.

Looking at the curves in Fig. 22, `mg-dwe` and `mg-dwe-diverse` generally behave alike. In terms of precision–recall curve, `mg-dwe` seems to have a slightly better precision at very low recall (probably with K at 1 or 2) but drops below `mg-dwe-diverse` at slightly higher recall.

Looking at recommendations using only the collaborative data provided by *Last.fm*, the results for `lfm-dwe` show that the recommendation accuracy becomes significantly worse. This is confirmed by a look at the ROC and precision–recall curves. The curve of `lfm-dwe` is significantly lower than the curve of `mg-dwe(-diverse)`. Recommenders with an underlying knowledge graph constructed with data from *MusicBrainz* know and are able to recommend all artists, even if they do not appear in the user history. This is different for the `lfm-dwe` recommender, which only uses collaborative data from *Last.fm* and only knows the artists that appear in user histories. This results in `lfm-dwe` never reaching a recall of 1. In terms of novelty and diversity, `lfm-dwe` performs slightly better than `mg-dwe(-diverse)`. Since there is a natural trade-off between accuracy and novelty and diversity, this result was to be expected.

It should be noted, that the better performance of `mg-dwe(-diverse)` compared to `lfm-dwe` does not necessarily implies that collaborative filtering is generally worse for artist recommendation than the method proposed by this work. A state-of-the-art collaborative filtering implementation would most likely perform better than `lfm-dwe`. However, comparing `mg-dwe(-diverse)` to `lfm-dwe` showed that adding domain knowledge to a simple bipartite item–user graph did improve the recommender systems performance significantly. The comparison of the evaluation results of `lfm-dwe` `mg-dwe(-diverse)` prove that EARS is not driven by the collaborative information in the knowledge graph.

Finally, the `rand-rec` recommender performs very poorly as expected. Novelty and diversity values are the best of the evaluated recommenders, which is also not surprising. Completely random recommendations naturally lead to more diverse and novel items, as similarity to artists a user has heard before is not considered at all.

6. Related work

This section provides an overview of different approaches to knowledge-based music recommendation. Particular emphasis is given to work that proposes music recommendation systems based on graph embeddings.

One of the first music recommender systems using external context information is *FOAFing the Music* (Celma & Serra, 2008). The system crawls thousands of websites for music-related information and filters that information to find music that fits a user's personal profile and listening habits. The music related information is stored as RDF triples and includes information regarding music releases, concerts, album reviews as well as users' listening histories from Last.fm. User profiles are represented by *Friend Of A Friend* (FOAF) profiles, which have to be created by the users themselves. The general recommendation approach is to obtain music-related information from a user's FOAF profile, identify the artists the user is interested in, and compute similar artist by exploiting the RDF graph of artists relationships. The author does not provide details regarding how exactly the similarity is computed, but it can be assumed that a simple path-based approach is used to find similar artist in the RDF graph.

Another path-based approach is the *dbrec* Music Recommendation System (Passant, 2010) buildt on DBpedia. The authors introduce *Linked Data Semantic Distance* (LDSD), which computes the similarity between two resources in DBpedia based on the direct and indirect links between them. Then a user is recommended such artists that have a small LDSD distance to her favorite artists. In addition, information is taken from DBpedia to provide explanations why two entities are similar, e.g. that two artists have the same birth place or associated labels. Explanations are basically formed by listing the facts both artists share in DBpedia. Although these explanations do not seem particularly sophisticated, *dbrec* was probably the first work to show that knowledge-based recommendation approaches can enable explainable recommendations.

In Oramas et al. (2017) knowledge graphs are used to recommend sounds for an online sound sharing platform. The approach extracts semantic features from textual descriptions of sounds, and applies entity

Table 8
Comparing related work with EARS.

Recommender	Music data	User data	Recommendation approach	Diversity	Explainability
(Celma 2008)	RDF graph created by web crawler	FOAF profiles	Path-based	–	–
(Passant, 2010)	RDF graph from DBpedia	–	Path-based	–	Shared DBpedia facts
(Oramas, 2017) sound recommender	RDF graph from DBpedia + WordNet	Downloaded sounds	Path-based and collaborative filter	–	–
(Chen, 2016) playlist recommender	–	KG from user playlists	HPE embedding (random walk)	–	–
(Wang, 2018) temporal preferences	Music meta data (no KG)	User playlists (no KG)	Direct embedding of playlists and music metadata	–	–
(Lin, 2018) temporal preferences	Collaborative KG (very simple)	Collaborative KG (playlists)	Embeddings (TRANS) and neural networks (HK-ANN)	Addressed	–
(Saravanou, 2021) playlist continuation	Spotify music data	Spotify playlist data	Aggregator functions to learn embeddings	–	–
EARS artist recommender	Collaborative KG from MusicBrainz	Collaborative KG from Last.fm	Arbitrary embedding method	Bounded Greedy Selection	KG-based

linking techniques to the extracted features using WordNet and DBpedia. Again, the recommendation process uses a path-based approach to create neighborhood vectors of items. These are combined with collaborative feature vectors that provide information about which sounds users have downloaded. This work can be classified as a hybrid method that combines knowledge graph-based recommendations and collaborative filtering. While a knowledge graph is used to generate neighborhood vectors of items, the collaborative information is not directly contained in the graph, but a feature combination is applied to consider the collaborative information.

Chen, Tsai et al. (2016) published one of the first works applying graph embedding techniques to music recommendations. Instead of recommending individual artists or songs, as in the previously discussed work, the authors proposed an approach in which music playlists are recommended as a whole. The underlying knowledge graph contains only the users and their playlists with the associated songs, but not other domain-specific information about the music tracks. The authors use their own embedding method called *Heterogenous Preference Embedding* (HPE) to embed the graph and make recommendations by computing the similarity between embeddings using simple distance measures. Generally speaking, HPE implements a random walk-based embedding technique to learn graph embeddings. However, this approach does not use a knowledge graph in the strict sense, since it lacks specific domain knowledge, but represents a bipartite user–item graph.

Wang et al. (2018) propose a context-aware music recommendation approach that can recommend music items that match users' temporal preferences for music. Unlike the other approaches discussed so far, they do not create a graph representing music knowledge, but propose to learn embeddings directly on user histories and music metadata. The authors assume that musical preferences stay stable within certain time periods. Based on this assumption, the proposed embedding technique treats the user's listening history as a temporal sequence. It considers songs that appear in a window of a certain size as similar and uses them to calculate song embeddings. Apart from that, the approach uses metadata for building feature vectors of songs. User preferences are represented by averaging embeddings of songs the user has listened to in the past. Recommended songs have embeddings that are similar to the embeddings of the user's preferences.

Lin et al. (2018) propose another approach that focuses on temporal aspects in music recommendations. Based on a user's current session, the system uses knowledge graph embeddings and recurrent neural networks to provide short-term recommendations for songs. The authors build a collaborative knowledge graph that includes songs, artists, albums, as well as users with their listening history, but that also lacks other domain-specific information about the music songs. Then they apply the distance-based embedding method `TransR` to obtain embeddings of songs and users. In addition to these graph embeddings, they compute embeddings of textual and visual data related to songs and aggregate the three resulting embeddings in a final one. The authors propose the *heterogeneous knowledge-based attentive neural network* (HK-ANN) that learns song recommendations and takes as input the user's embeddings and the embeddings of songs they have recently used. Their evaluation shows that the proposed system is able to outperform other short-term music recommendation systems. In addition, more unpopular songs are recommended compared to other approaches, indicating that the approach is less prone to popularity bias.

`MUSIG` is another knowledge graph embedding-based music recommender system that was recently proposed (Saravanou et al., 2021). `MUSIG` makes use of the data available in Spotify and considers three different types of information: organizational information such as co-occurrences in playlists, content information based on audio files and expert generated information such as genres. Rather than learning different embedding vectors for individual entities in the graph, `MUSIG` learns aggregator functions that can compute embeddings for a given node by looking at information in the node's neighborhood. Then a *multi-task supervision step* learns the parameter of the aggregation functions, which can be used for the following tasks: playlist prediction, genre prediction and audio similarity prediction. The authors applied their method to a playlist continuation task. They calculate the average embedding of songs in a playlist and compute cosine similarities between this average embedding and the embeddings of songs not in the playlist. This approach is able to calculate embeddings of new items very efficiently by just applying the average functions to the neighborhood of the new item.

Table 8 illustrates the differences between the above mentioned works and compares them with the EARS system. It indicates that EARS provides a flexible approach based entirely on embedding knowledge graphs. The collaborative knowledge graph combines semantically rich information about artists with collaborative user data. Moreover, EARS explicitly addresses recommendation diversity and explainability.

7. Conclusion

In this paper we have presented the EARS recommender approach, which is based on knowledge graph embeddings. The underlying knowledge graph is built from open data about the music domain using *MusicBrainz* and *Last.fm*. We followed a hybrid approach to derive a knowledge graph that combines relevant domain knowledge about music artists and bands with collaborative data about users' listening behavior. In addition, EARS puts special emphasis on the diversity of recommendations and their explainability.

A comprehensive evaluation with real user data from *Last.fm* proves that EARS delivers very good recommendation results. In particular, it was shown that the use of knowledge graph embeddings can significantly improve the recommendation quality of purely collaborative approaches. Furthermore, a comprehensive set of experiments investigated the influence of different hyper parameters of `DeepWalk`, which turned out as the embedding method showing the best recommendation results.

As future avenues we envision building a more sophisticated knowledge graph and evaluating how it affects the recommendations. Moreover, to our knowledge, no openly accessible knowledge graph exists in the music domain yet.

Regarding the recommender system, it would also be interesting to study directed or weighted knowledge graphs. For example, relations between users and artists could be weighted by how often a user has played the artist in the past. Similarly, weights between labels and artists could be based on how many times a label has published an artist's release.

So far, we have only compared a hybrid knowledge graph of *MusicBrainz* and *Last.fm* data with a knowledge graph of purely collaborative *Last.fm* data. Of course, it would also be interesting to see how a recommender system that uses only domain-specific knowledge from *MusicBrainz* and omits collaborative knowledge compares to the hybrid approach taken by EARS.

CRediT authorship contribution statement

Niels Bertram: Conceptualization, Methodology, Software, Investigation, Writing – original draft. **Jürgen Dunkel:** Conceptualization, Methodology, Investigation, Writing – original draft. **Ramón Hermoso:** Conceptualization, Methodology, Investigation, Writing – original draft.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ramon Hermoso reports equipment, drugs, or supplies and travel were provided by Spain Ministry of Science and Innovation. Ramon Hermoso reports a relationship with Spain Ministry of Science and Innovation that includes: funding grants.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by project PID2020-113037RB-I00, funded by MCIN/AEI/10.13039/501100011033 and the Government of Aragon (Group Reference T64_20R and T64_23R, COSMOS research group).

References

Abdollahpouri, H., Burke, R., & Mobasher, B. (2019). Managing popularity bias in recommender systems with personalized re-ranking. In *The thirty-second international flairs conference*.

Aggarwal, C. C. (2016). *Recommender systems: the textbook* (1st ed.). Cham: Springer International Publishing: Imprint: Springer, http://dx.doi.org/10.1007/978-3-319-29659-3.

Belkin, M., & Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, *14*.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517. http://dx.doi.org/10.1145/361002.361007.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems*, *26*, 9.

Brackett, D. (2016). *Categorizing sound: genre and twentieth-century popular music* (1st ed.). University of California Press, http://dx.doi.org/10.1525/california/9780520248717.001.0001.

Cao, S., Lu, W., & Xu, Q. (2015). GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management* (pp. 891–900). Melbourne Australia: ACM, http://dx.doi.org/10.1145/2806416.2806512.

Celma, Ò. (2010). *Music recommendation and discovery*. Berlin, Heidelberg: Springer Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-642-13287-2.

Celma, Ò., & Serra, X. (2008). Foafing the music: Bridging the semantic gap in music recommendation. *Journal of Web Semantics*, *6*(4), 250–256.

Chen, C.-M., Tsai, M.-F., Lin, Y.-C., & Yang, Y.-H. (2016). Query-based music recommendations via preference embedding. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 79–82). Boston Massachusetts USA: ACM, http://dx.doi.org/10.1145/2959100.2959169.

Chen, C.-M., Yang, C.-Y., Hsia, C.-C., Chen, Y., & Tsai, M.-F. (2016). Music playlist recommendation via preference embedding. (p. 2).

Chicaiza, J., & Valdiviezo-Diaz, P. (2021). A comprehensive survey of knowledge graph-based recommender systems: Technologies, development, and contributions. *Information*, *12*(6), 232. http://dx.doi.org/10.3390/info12060232.

Dettmers, T., Minervini, P., Stenetorp, P., & Riedel, S. (2018). Convolutional 2D knowledge graph embeddings. *32*, In *Proceedings of the AAAI conference on artificial intelligence* (p. 8).

Friedlander, J. P. (2015). News and notes on 2015 RIAA shipment and revenue statistics. (p. 3).

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78–94.

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable Feature Learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 855–864). San Francisco California USA: ACM, http://dx.doi.org/10.1145/2939672.2939754.

Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., & He, Q. (2020). A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering*.

Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2022). Knowledge graphs. *ACM Computing Surveys*, *54*(4), 1–37. http://dx.doi.org/10.1145/3447772.

Ji, S., Pan, S., Cambria, E., Marttinen, P., & Philip, S. Y. (2021). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(2), 494–514.

Jia, B., Huang, X., & Jiao, S. (2018). Application of semantic similarity calculation based on knowledge graph for personalized study recommendation service. *Educational Sciences: Theory & Practice*, *18*(6), http://dx.doi.org/10.12738/estp.2018.6.195.

Kowald, D., Schedl, M., & Lex, E. (2020). The unfairness of popularity bias in music recommendation: a reproducibility study. In J. M. Jose, E. Yilmaz, J. a. Magalhães, P. Castells, N. Ferro, M. J. Silva, & F. Martins (Eds.), *Advances in information retrieval, Vol. 12036* (pp. 35–42). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-45442-5_5.

Lena, J. C., & Peterson, R. A. (2008). Classification as culture: Types and trajectories of music genres. *American Sociological Review*, *73*(5), 697–718. http://dx.doi.org/10.1177/000312240807300501.

Li, J., Wu, L., Guo, R., Liu, C., & Liu, H. (2019). Multi-level network embedding with boosted low-rank matrix approximation. In *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining* (pp. 49–56).

Lin, Q., Niu, Y., Zhu, Y., Lu, H., Mushonga, K. Z., & Niu, Z. (2018). Heterogeneous knowledge-based attentive neural networks for short-term music recommendations. *IEEE Access*, *6*, 58990–59000. http://dx.doi.org/10.1109/ACCESS.2018.2874959.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. http://dx.doi.org/10.48550/ARXIV.1310.4546.

Morris, J. W. (2012). Making music behave: Metadata and the digital music commodity. *New Media & Society*, *14*(5), 850–866. http://dx.doi.org/10.1177/1461444811430645.

Oramas, S., Ostuni, V. C., Noia, T. D., Serra, X., & Sciascio, E. D. (2017). Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology*, *8*(2), 1–21. http://dx.doi.org/10.1145/2926718.

Passant, A. (2010). Dbrec — music recommendations using dbpedia. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, & B. Glimm (Eds.), *The semantic web – ISWC 2010, Vol. 6497* (pp. 209–224). Berlin, Heidelberg: Springer Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-642-17749-1_14.

Paulheim, H. (2016). Knowledge graph refinement: A survey of approaches and evaluation methods. In P. Cimiano (Ed.), *Semantic Web*, *8*(3), 489–508. http://dx.doi.org/10.3233/SW-160218.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 701–710).

Perozzi, B., Kulkarni, V., Chen, H., Skiena, S., & Don't Walk, S. (2017). Online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM international conference on advances in social networks analysis and mining* (pp. 258–265).

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., & Tang, J. (2018). Network embedding as matrix factorization: Unifying deepwalk, LINE, PTE, and node2vec. In *Proceedings of the Eleventh ACM international conference on web search and data mining* (pp. 459–467).

Rozemberczki, B., & Sarkar, R. (2018). Fast sequence-based embedding with diffusion graphs. In *International workshop on complex networks* (pp. 99–107). Springer.

Saravanou, A., Tomasi, F., Mehrotra, R., & Lalmas, M. (2021). Multi-task learning of graph-based inductive representations of music content. (p. 8).

Schedl, M., Zamani, H., Chen, C.-W., Deldjoo, Y., & Elahi, M. (2018). Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, *7*(2), 95–116. http://dx.doi.org/10.1007/s13735-018-0154-2.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European semantic web conference* (pp. 593–607). Springer.

Shi, C., Zhang, Z., Luo, P., Yu, P. S., Yue, Y., & Wu, B. (2015). Semantic path based personalized recommendation on weighted heterogeneous information networks. In *Proceedings of the 24th ACM international on conference on information and knowledge management* (pp. 453–462). Melbourne Australia: ACM, http://dx.doi.org/10.1145/2806416.2806528.

Silveira, T., Zhang, M., Lin, X., Liu, Y., & Ma, S. (2019). How good your recommender system is? A survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*, *10*(5), 813–831. http://dx.doi.org/10.1007/s13042-017-0762-9.

Smyth, B., & McClave, P. (2001). Similarity vs. Diversity. In G. Goos, J. Hartmanis, J. van Leeuwen, D. W. Aha, & I. Watson (Eds.), *Case-based reasoning research and development, Vol. 2080* (pp. 347–361). Berlin, Heidelberg: Springer Berlin Heidelberg, http://dx.doi.org/10.1007/3-540-44593-5_25.

Sun, D. L., & Fevotte, C. (2014). Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 6201–6205). Florence, Italy: IEEE, http://dx.doi.org/10.1109/ICASSP.2014.6854796.

Sun, Y., Han, J., Yan, X., Yu, P. S., & Wu, T. (2011). PathSim: Meta path-based top-K similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, *4*(11), 992–1003. http://dx.doi.org/10.14778/3402707.3402736.

Torres, L., Chan, K. S., & Eliassi-Rad, T. (2020). GLEE: Geometric Laplacian eigenmap embedding. *Journal of Complex Networks*, *8*(2), cnaa007.

Wang, D., Deng, S., Zhang, X., & Xu, G. (2018). Learning to embed music and metadata for context-aware music recommendation. *World Wide Web*, *21*(5), 1399–1423. http://dx.doi.org/10.1007/s11280-017-0521-6.

Wang, X., He, X., Cao, Y., Liu, M., & Chua, T.-S. (2019). KGAT: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 950–958).

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A Survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*(12), 2724–2743. http://dx.doi.org/10.1109/TKDE.2017.2754499.

Werner, A. (2020). Organizing music, organizing gender: algorithmic culture and Spotify recommendations. *Popular Communication*, *18*(1), 78–90.

Yang, D., Rosso, P., Li, B., & Cudre-Mauroux, P. (2019). NodeSketch: Highly-Efficient Graph Embeddings via recursive sketching. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 1162–1172). Anchorage AK USA: ACM, http://dx.doi.org/10.1145/3292500.3330951.

Zhang, Z., Cui, P., Li, H., Wang, X., & Zhu, W. (2018). Billion-scale network embedding with iterative random projection. In *2018 IEEE international conference on data mining (ICDM)* (pp. 787–796). Singapore: IEEE, http://dx.doi.org/10.1109/ICDM.2018.00094.

Zhang, M., & Hurley, N. (2009). Novel item recommendation by user profile partitioning. In *2009 IEEE/WIC/ACM international joint conference on web intelligence and intelligent agent technology, Vol. 1* (pp. 508–515). http://dx.doi.org/10.1109/WI-IAT.2009.85.

Ziegler, C.-N., McNee, S. M., Konstan, J. A., & Lausen, G. (2005). Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on world wide web* (pp. 22–32).