Random Walk with Restart for Automatic Playlist Continuation and Query-Specific Adaptations

Timo van Niedek
Radboud University
Nijmegen, The Netherlands
timo.niedek@science.ru.nl

Arjen P. de Vries
Radboud University
Nijmegen, The Netherlands
arjen@acm.org

ABSTRACT

This Recsys Challenge paper by Team Radboud presents a solution to the automatic playlist continuation (APC) task using random walks, inspired by Pinterest's Pixie [5] and earlier work by the second author [4]. The generic idea of recommendation using random walks is specialised to the APC task by the specific choices made to represent playlists and tracks as a graph and by the design of pruning methods that reduce the scope of the random walks. Playlist characteristics are captured through track metadata, audio features and playlist title. Scoping of the random walks considers 1) title-based prefiltering, 2) title-based retrieval, 3) feature-based prefiltering and 4) degree pruning. Our methods are effective in reducing popularity bias, a common issue in related methods. The best performing recommender uses a hybrid degree pruning techinque, for which we report an R-Precision of 0.1982, an NDCG of 0.3564 and a Recommended Songs Clicks of 2.2934, achieving 21st place on the main leaderboard and 9th on the creative leaderboard of the RecSys Challenge 2018. For a more complete description of the approach and evaluation results we refer to the first author's Master's Thesis [14].

CCS CONCEPTS

• **Information systems** → **Music retrieval**; *Recommender systems*;

KEYWORDS

Automatic Playlist Continuation, Random Walk with Restart, Prefiltering

ACM Reference Format:
Timo van Niedek and Arjen P. de Vries. 2018. Random Walk with Restart for Automatic Playlist Continuation and Query-Specific Adaptations. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18), October 2, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3267471.3267483

RecSys Challenge '18, October 2, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-6586-4/18/10…$15.00
https://doi.org/10.1145/3267471.3267483

1 INTRODUCTION

The Team Radboud approach to Automatic Playlist Continuation (APC) in the ACM RecSys Challenge 2018 [3] is built on the assumption formulated in Lee et al. [8]: playlists are created with a specific purpose, and with this assumption purpose is reflected in measurable characteristics of its tracks. Recommendations should then identify similar tracks that match these characteristics.

Intuitively, tracks that appear in the same playlists are likely similar. Likewise, playlists that have one or more tracks in common are similar as well. Random walks over the playlist graph therefore constitute a measure of similarity between tracks and playlists. We generate recommendations using random walks over a bipartite graph representation of tracks and playlists, inspired by the Pixie recommender system by Eksombatchai et al. [5][1].

We apply *contextual prefiltering* [2] to constrain the walks to the most relevant tracks and playlists, leaving the specification of context implicit in the features selected to represent playlists and tracks. Insights from [10] suggest to represent playlists by their title. We further consider track metadata and content (using features obtained from the Spotify API[2]). Finally, we attempt to reduce the effect of the popularity bias using degree pruning.

2 DATA

We report effectiveness of the various algorithms on the *Million Playlist Dataset* (MPD) [1]. We represent tracks by their *release year*, their *explicit* label, and their audio features; a description of the latter provided in the Spotify API documentation[3]. We use *danceability*, *energy*, *speechiness*, *acousticness*, *instrumentalness*, *liveness*, *valence* and *tempo*. We add a new feature *count*, equal to the logarithm of the track frequency, normalized by the logarithm of the maximum track frequency in the MPD.

3 METHODS

We represent the MPD as a bipartite graph, which we call the *playlist graph*. The playlist graph is the main data structure used throughout this work, as track recommendations are the result of multiple random walks over this graph. We additionally use playlist titles for prefiltering and ranking titles to address cold-start. Furthermore, we introduce playlist models estimated from the metadata for prefiltering, and prune the graph by degree to address the popularity bias.

[1] Created for image sharing platform Pinterest https://www.pinterest.com
[2] https://developer.spotify.com/documentation/web-api/reference
[3] https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features

3.1 The Playlist Graph

The playlist graph is an undirected bipartite graph $G = (T, P, E)$ that connects tracks T with playlists P, using undirected edges E. Existence of an edge (t, p) represents the fact that track $t \in T$ is included in playlist $p \in P$. We allow the existence of multiple edges between a playlist node p and a track t in order to model cases where a user has included a track multiple times in their playlist.

For efficient implementation of neighbor sampling during the random walks, we follow [5] and represent the entire playlist graph (consisting of 1.000.000 playlists and 2.262.292 unique tracks) using adjacency lists, as follows. Assign node IDs in the range of 0 to $|P|$ to the playlists, and IDs in the range of $|P|$ to $|P| + |T|$ to the unique tracks ($|P| = 1.000.000$, $|T| = 2.262.292$). Compute the adjacency lists for all of the $|P| + |T|$ nodes by iterating over the MPD and concatenate all adjacency lists into a single adjacency vector **a**. The indices at which to find the start of the adjacency list for a given node are stored in the offsets vector **o**. The neighborhood operation then corresponds to a constant time list slice as $N_G(v) = \{a_i, a_{i+1}, \ldots, a_{j-1}\}$, where $i = o_v, j = o_{v+1}$. The neighborhood of a playlist node consists of the tracks included in that playlist, and the neighborhood of a track node corresponds to all playlists in which that track was included. The degree of a node is computed using $\deg(v) = o_{v+1} - o_v$. The resulting graph consumes ~ 1.8 GB of memory, i.e., it fits in the memory of a single machine.

3.2 Random Walks

3.2.1 Single Random Walk. The single random walk, shown in Algorithm 3.1, generates recommendations with a single query node as input by approximating its soft neighborhood (see also [13]). The transition probabilities are uniform. The random walk starts by initializing the visit counts for every track node to 0. The outer loop runs a number of walks until the maximum number of steps N is reached. At the start of each walk, the current node is reset to the query node q. Each step within the walk consists of three operations [5]:

(1) Given the current track t_{curr} (initially set to the query track q), sample a playlist p_{curr} from its neighbors $N_G(t_{curr})$.
(2) Given p_{curr}, sample a track t'_{curr} from its neighbors $N_G(p_{curr})$.
(3) Update the current track t_{curr} to t'_{curr} and repeat.

The RandomGeometric function samples a walk length from the geometric distribution with non-zero restart probability α. If $\alpha = 0$ the walk will never restart; if $\alpha = 1$ the walk will restart after a single step consisting of two edge traversals, a model equivalent to the Random-Random model described by Leskovec et al. [9]. The result of the single random walk is a dictionary of (node: visit count) pairs, where the nodes with highest visit count are most similar to the query node.

3.2.2 Multiple Random Walks. To generate recommendations for a playlist, we run the single random walk for every query node and aggregate the results (Algorithm 3.2). The differences from [5] are that N_q is assigned uniformly, and multi-hit boosting is not applied. Instead, to obtain a ranking over tracks based on the query playlist, we simply sum the visit counts for all query nodes.

The single random walks are completely independent of each other. We can therefore parallelize the for-loop in Algorithm 3.2 to speed up computation time by utilizing multiple CPUs. Because the random walk algorithms do not require write access to the playlist graph, the graph can reside in shared memory and no locking is required. Parallelization of the random walks leads to a speedup roughly by a factor of 2.3 when utilizing five processes.

Algorithm 3.1 Single Random Walk with Restart (based on [5])

```
function SingleRandomWalk(q: Query Node, G = (T, P, E): Playlist Graph, α: Restart Probability, N_q)
    n ← 0
    V ← {v: 0} ∀_{v∈T}
    repeat
        t_curr ← q
        n_curr ← 0
        N_curr ← RandomGeometric(α)
        repeat
            p_curr ← RandomNeighbor(G, t_curr)
            t_curr ← RandomNeighbor(G, p_curr)
            V[t_curr] ← V[t_curr] + 1
            n_curr ← n_curr + 1
        until (n_curr ≥ N_curr) ∨ (n + n_curr ≥ N_q)
        n ← n + n_curr
    until n ≥ N_q
    return V
end function
```

Algorithm 3.2 Multiple Random Walks with Restart (based on [5])

```
function MultipleRandomWalk(q: Query Tracks, G: Playlist Graph, α: Restart Probability, N)
    N_q ← N/|q|
    V ← {v: 0} ∀_{v∈T}
    for all q ∈ q do
        V_q ← SingleRandomWalk(q, G, α, N_q)
        V ← V + V_q
    end for
    return V
end function
```

3.3 Processing Titles

Playlist titles may capture their creator's intent [12]. E.g., clustering playlists by their title into contextual clusters, and constraining collaborative filtering to those clusters, has been shown to lead to large increases in music recommendation effectiveness [11]. We use titles in contextual prefiltering and to address cold-start. Titles are preprocessed following the steps described in [14].

3.3.1 Title-based Prefiltering. For a given query playlist, we filter the graph keeping only the playlists whose titles contain at least all the terms in the query playlist. We apply title-based prefiltering only if the graph after filtering contains more than 150 playlist nodes, as title-based prefiltering is only useful for titles that imply some shared meaning, not when the title is only meaningful to the creator of the playlist.

3.3.2 Title-based Retrieval. A subset of all playlists is empty: the user has determined a playlist title but not yet added any tracks.

This setting can be approached using a metadata-based query [12]. We select playlists with a BM25 score of at least 5.0 to find the most commonly occurring tracks in these playlists. We reduce the playlist graph to include only the tracks appearing in at least 15% of the retrieved playlists, limiting the result to 100 tracks only (for efficiency reasons).

3.4 Modeling Playlist Content

While the random walks compute similarity scores between tracks based on the local graph structure, they are agnostic of the content-based similarity between tracks or playlists. To improve the coherence of the recommendations with respect to the query playlist, we propose to model the contents of a playlist using content-based features and metadata for each track. We use these models for prefiltering and graph pruning.

3.4.1 Histogram Model. A track t is represented by the features described in Section 2. We define a playlist model by an independent feature distribution $P(f_i|p)$ for each dimension i. We use normalized histograms as an estimate of the continuous probability distribution function, without making assumptions about the modality or skewness of the underlying distributions. The feature distribution for a feature i is estimated from its empirical distribution as $P(f_i \in b|p) = \frac{H(b)}{N_i}$ where b is the bin in which the value f_i falls, $H(b)$ is the number of tracks in playlist p that fall in bin b and N_i is the total number of observations for feature i in playlist p, not counting missing values. To address sparseness of the histogram distribution derived for short playlists, we use Jelinek-Mercer smoothing [7] with the collection model $P(f_i|P)$.

3.4.2 Feature-based Prefiltering. We argue that the playlist model encodes playlist purpose information [14] and therefore should be used to determine a subgraph of playlists with low distance to the query playlist, removing the remaining playlists at runtime. This concentrates the random walks to playlists that are more likely to be relevant, and recommends tracks that fit the distribution of the query playlist better.

Considering runtime efficiency costs, we use the ℓ_1 distance between playlist histograms. Playlists with an ℓ_1 distance larger than a threshold θ are discarded from the graph. To reduce the risk of thinning the graph too much, we only apply feature-based prefiltering if the resulting graph contains at least 150 playlists. An additional criterion for filtering is that the query playlist contains at least five tracks, for the simple reason that the histogram model of a very short playlist is not informative.

3.5 Degree Pruning

High-degree nodes consume memory while they diffuse the random walks over a very large number of nodes, possibly counterproductive when using the walk to define a measure of similarity over the graph. The Pixie recommender [5] therefore removes edges for high-degree nodes. Since we observe a skewed heavy-tailed distribution for tracks [14], we introduce a similar approach. We rank the neighboring playlists based on the likelihood of drawing the track t from the playlist p. Since a track is defined as a set of independent feature values f_i for feature dimensions i, the track log likelihood equals $\log P(t|p) = \sum_{i=1}^{|F|} \log P(f_i|p)$, where missing values for a track t are simply ignored. The playlists are sorted based on the log-likelihood, and edges between t and playlists with low log-likelihood are removed, until the number of edges equals $\deg(t)^{\delta}$. The amount of edges that are removed is determined by pruning factor δ ($0 \leq \delta \leq 1$).

Table 1: Performance stratified on the number of query tracks using $\alpha = 0.99$, $N = 50000$ and title-based retrieval for the first row. We consider this recommender to be the baseline approach.

# Query Tracks	R-Precision	NDCG	RS Clicks
0	0.1019	0.2130	11.5532
1	0.1366	0.2815	4.0090
5	0.1818	0.3618	2.0345
10	0.1961	0.3924	1.1400
25	0.2047	0.4068	0.6595
100	0.2115	0.4147	0.2445
Average	0.1827	0.3646	2.3682

4 EVALUATION

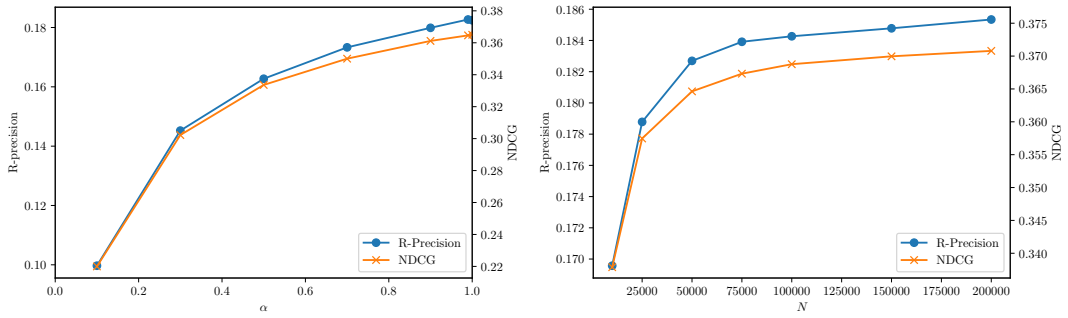
4.1 Experimental Setup and Metrics

To reduce the cost of experiments, we create a validation set following the structure of the challenge set. We sample 1000 playlists from the MPD per challenge category, giving a validation set of 10,000 playlists in total. The playlists are sampled to ensure that the number of held-out tracks is at least half of the number of query tracks when splitting the playlists. For every playlist, the random walk is run with the query tracks as input. We ensure that the held-out tracks are hidden by disabling all edges from and to the query playlist node. A set of 500 recommended tracks is generated from the query tracks, and compared to the held-out tracks. Recommendations are scored using the R-Precision, NDCG and *Recommended Songs Clicks* (RS Clicks) metrics (refer to [3] for definitions); we report the mean scores over the validation playlists.

4.2 Multiple Random Walks

First, we evaluate the performance of multiple random walks with restart over the full playlist graph. Parameters of Algorithm 3.2 are the restart probability α and the number of steps per playlist N. Surprisingly, performance in R-Precision and NDCG (Fig. 1a) is best for high restart probabilities, when the random walk is reduced to a single step (track → playlist → track) from the starting node [9]. Fig. 1b shows recommendation effectiveness against the number of steps. Recommendation effectiveness increases with the number of steps. Unfortunately, runtime performance scales linearly with the number of steps [5]. The performance gains for more than 50000 steps are relatively small, so we run the rest of our experiments using $N = 50000$.

Experiments using RS Clicks follow these trends: higher values for α and a higher number of steps lead to a lower number of clicks. RS Clicks equals 2.3682 for $\alpha = 0.99, N = 50000$.



(a) (b)

Figure 1: (a) Performance of the random walks against the restart probability α with $N = 50000$. (b) Performance of the random walks against the number of steps N with $\alpha = 0.99$. We omit the recommended songs clicks for visual clarity.

Table 2: Performance of different recommenders: (1) random walk with title-based retrieval for 0-length query playlists, (2) the same recommender with title-based prefiltering for all playlists and (3) a hybrid recommender combining the best results of (1) and (2).

Recommender	R-Precision	NDCG	RS Clicks
(1) Baseline	0.1827	0.3646	2.3682
(2) All prefiltered	0.1650	0.3301	3.7993
(3) Hybrid	0.1858	0.3671	2.3042

4.3 Title-based Retrieval

One tenth of the challenge set consists of cold-start playlists with just a title [3], for which we expect low performance: playlist titles are noisy and provide limited information. We evaluate the title-based retrieval system described in Section 3.3.2 on the subset of 1000 cold-start validation playlists. Random walks start from the results of title-based retrieval, using $\alpha = 0.99, N = 50000$. Performance measurements are summarised in the first row of Table 1. Additionally, we observe that playlist titles that correspond to soundtracks of movies, series, a specific artist or album perform very well (see [14]).

4.4 Title-based Prefiltering

We evaluate title-based prefiltering using an experimental setup similar to Pichl et al. [10], and evaluate three different recommenders: (1) a baseline approach using the random walk described in the previous section, with title-based retrieval for the zero-length playlists, (2) the same recommender with title-based prefiltering applied for every query playlist, and (3) a hybrid or switching recommender. For the hybrid recommender, we compute R-precision, NDCG and RS Clicks of recommender (1) and (2), per title. Given a query playlist, we switch between these two recommenders based on majority voting: we choose the recommender that wins for at least two out of three metrics.

Table 2 shows a comparison of the performance of the three recommenders on the validation set. The baseline recommender (1) outperforms recommender (2) where prefiltering is applied to every playlist, by a large margin for all three metrics. The hybrid recommender (3) outperforms both methods, but only by a small margin for the baseline recommender.

Table 3: Performance of recommenders that include feature-based prefiltering. Whenever Jelinek-Mercer smoothing is used, we set $\lambda = 0.25$. The threshold θ determines the amount of prefiltering.

Recommender	R-Precision	NDCG	RS Clicks
Without smoothing			
$\theta = 4$	**0.1836**	**0.3655**	2.4047
$\theta = 5$	0.1825	0.3644	2.3593
$\theta = 7$	0.1812	0.3627	2.4428
$\theta = 10$	0.1826	0.3650	2.3695
With smoothing			
$\theta = 4$	0.1800	0.3596	2.3964
$\theta = 5$	0.1773	0.3552	2.5126
$\theta = 7$	0.1822	0.3640	2.4276
$\theta = 10$	0.1826	0.3647	2.3763
Baseline	0.1827	0.3646	**2.3682**

4.5 Feature-based Prefiltering

We now turn to evaluation of the feature-based prefiltering method (Section 3.4.2), varying the amount of smoothing λ and pruning threshold θ. We choose between $\lambda = 0$ and $\lambda = 0.25$ to determine whether smoothing is useful. The values for θ were chosen by ranking the collection for a random sample of 100 playlists as described in [14]. Table 3 shows the results, finding the best performance without smoothing and $\theta = 4$.

4.6 Degree Pruning

We finally evaluate the performance of graph cleaning by pruning edges from track nodes with high degree as described in Section 3.5. The amount of pruning is controlled by pruning factor δ, where

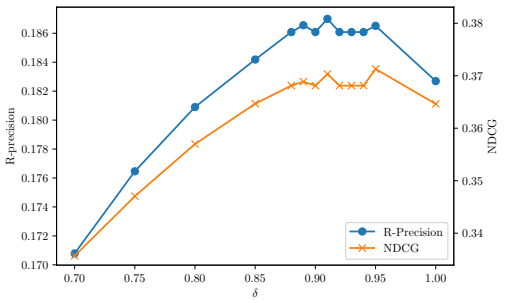


Figure 2: Performance of the random walks against the pruning factor δ.

Table 4: Performance of different recommenders: (1) random walk with title-based retrieval for 0-length query playlists, (2) the same recommender with degree pruning factor $\delta = 0.95$ and (3) a hybrid recommender switching between δ's following Eq. (1).

Recommender	R-Precision	NDCG	RS Clicks
(1) Baseline	0.1827	0.3646	2.3682
(2) $\delta = 0.95$	0.1865	0.3712	2.3585
(3) Hybrid	0.1913	0.3770	2.3375

$\delta = 1.0$ corresponds to a random walk without degree pruning. Lower values of δ correspond to a larger amount of edges pruned.

Fig. 2 shows the performance of the random walk for different values of δ. Degree pruning leads to an improved performance in terms of R-Precision and NDCG. For R-Precision, the score peaks at $\delta = 0.91$ and for NDCG at $\delta = 0.95$. The RS Clicks metric is lowest at $\delta = 0.95$ with a value of 2.3585. We additionally found that degree pruning only leads to an improved performance when the set of query tracks is sufficiently large [14]. For smaller query playlists, it is useful to prune less, or not at all. Based on these findings, we propose a hybrid switching recommender that switches between different values of δ based on the number of query tracks $|\mathbf{q}|$:

$$\delta = \begin{cases} 0.75 & \text{if } |\mathbf{q}| \geq 100 \\ 0.9 & \text{if } 25 \leq |\mathbf{q}| < 100 \\ 0.95 & \text{if } 5 \leq |\mathbf{q}| < 25 \\ 1.0 & \text{if } |\mathbf{q}| < 5 \end{cases} \quad (1)$$

Table 4 summarizes the results of degree pruning. The hybrid switching recommender improves upon the baseline and upon degree pruning with a single value of $\delta = 0.95$.

5 RESULTS

Table 5 summarises the best performing recommenders. We test the significance of the difference in performance between the baseline and the other recommenders with a Wilcoxon signed-rank test since the scores were non-normal. All proposed extensions lead to an increase in performance over the baseline system in terms of R-Precision and NDCG. The same holds for RS Clicks, except for feature-based prefiltering. For the first two metrics all differences were significant. The hybrid degree pruning recommender gave the biggest increase in performance compared to the baseline.

Table 5: Summary of the performance of different recommenders on the validation set. Significance of the difference between the baseline performance and others is tested using a Wilcoxon signed-rank test.

Recommender	R-Precision	NDCG	RS Clicks
Baseline	0.1827	0.3646	2.3682
Hybrid title-based pf.	0.1858†	0.3671†	**2.3042**
Feature-based pf.	0.1836*	0.3655†	2.4047
Degree pruning	0.1865†	0.3712†	2.3585*
Hybrid degree pruning	**0.1913**†	**0.3770**†	2.3375

* Significant with $p < 0.05$

† Significant with $p < 0.01$

Table 6: Summary of the performance of different recommenders on the challenge set. The number of steps N is doubled to 100,000 to improve performance for the challenge.

Recommender	R-Precision	NDCG	RS Clicks
Baseline*	0.1956	0.3535	**2.2566**
Degree pruning*	0.1982	0.3569	2.3442
Hybrid degree pruning*	**0.1985**	**0.3571**	2.3328
1. Main track winner	0.2241	0.3946	1.7839
21. Baseline	0.1954	0.3521	2.2946
1. Creative track winner	0.2234	0.3939	1.7845
9. Hybrid degree pruning	0.1982	0.3564	2.2934

* Evaluated on 50% of the challenge set

Table 6 shows results on the challenge set. Due to time constraints, we were unable to evaluate all systems on the challenge set, and focus on the baseline, degree pruning, and hybrid degree pruning. The scores reported are taken from the challenge dashboard, which used 50% of the challenge set for evaluation. The hybrid degree pruning recommender performed best on the 50% challenge set in terms of the R-Precision and NDCG. The baseline system outperformed both degree pruning recommenders in terms of the RS Clicks. As a precaution against overfitting on the challenge set, the score of the latest submission was recalculated on the full test set after the challenge closed.

6 DISCUSSION

Key strengths of the proposed method are the flexibility of integrating various types of evidence to establish similarity between tracks and playlists, and scalability to large graphs.

Our baseline recommender performed well on the 50% challenge set, with minor improvements using the proposed extensions. Most relevant tracks seem to lie within the two-hop neighborhood of a query track. High values for the restart probability α lead to the best performance; long random walks are likely to diverge quickly, and lead to a strong popularity bias, since high-degree nodes are

much more likely to be visited. A more extensive analysis following Jannach et al. [6] in Van Niedek's MSc thesis [14] supports this hypothesis empirically, showing further that with explicit lyrics and/or strong electronic beats are overrepresented in the random walks when compared to the query playlists.

From the title-based prefiltering experiments, we conclude that a large majority of the playlist titles are uninformative or simply do not share the same intent even if they share title terms. Feature-based prefiltering led to a (negligible) increase in performance, for a relatively low threshold. The playlists that met the criterion of 150 remaining playlists (of which there were very few) did however benefit from feature-based prefiltering. Feature-based prefiltering is thus promising in specific cases. The most useful extension turned out to be the degree pruning, likely due to the diffusion of random walks through high degree nodes. Longer query playlists benefit from pruning, where summing visit counts over a large number of input tracks compensates for the increased sparsity in the resulting graph. Short query playlists should however not be pruned, for too many relevant edges would be removed from the neighborhood.

Finally, we discuss a few shortcomings of our work. Even though the runtime of the random walk is constant with respect to the graph size, a single run over the validation set still takes approximately 8-9 hours. To save computation time, we opted to use a relatively simple playlist model and distance metric, while more sophisticated models might lead to stronger improvements. Additionally, some combinations of parameters were left untested. Specifically, we were unable to combine the different recommenders into a system that incorporates all extensions. Finally, we did not evaluate all systems on the challenge set. In spite of the recognized limitations, our experiments should provide a solid understanding of the challenges and possible solutions for automatic playlist continuation.

7 CONCLUSION

We defined the bipartite playlist graph for automatic playlist continuation (APC), an efficient data structure for representing a large collection of playlists and tracks. We adapted the previously proposed multiple random walk with restart algorithm and found that extremely short walks with only a single step on average perform best. Preliminary scores on the challenge set showed that the random walk with restart, which we used as a baseline, already achieves a competitive performance in the context of the challenge. The proposed methods run in constant time with respect to the graph size, can be parallelized easily and are flexible and extensible.

We proposed title-only retrieval to address the cold-start scenario. We investigated the effectiveness of title and feature-based prefiltering. Both methods only led to small improvements upon the baseline. We then used a pruning method based on the distance between tracks and playlists in terms of the metadata and features, which yielded a sizable increase in performance above the baseline. We additionally observed that for short query playlists, little to no pruning is better, while long playlists benefited from heavy pruning. The best performance reported in this work was achieved by combining multiple levels of pruning into a hybrid recommender; this system achieved an R-Precision of 0.1982, an NCDG of 0.3564 and a Recommended Songs Clicks of 2.2934 on the challenge set, resulting in a 9th place on the creative leaderboard of the RecSys Challenge 2018 [3].

Directions for future work include combining the different extensions proposed, in order to see if performance can be improved further. Additionally, one can think of many more extensions that could combine the graph structure with metadata and audio features, such as biasing the random walk using weighted transitions, or adding new edges between playlists based on the track features or titles. A major hindrance for random walks is popularity bias. While we have proposed methods that reduce this bias, we believe that more insight can be gained using user-centric metrics or metrics that measure the coherence of recommendations. Our results show that random walk with restart using query-specific adaptations is a promising new direction for the domain of automatic playlist continuation.

This research was partially funded by NWO grant 628.011.001 (SQIREL-GRAPHS). The code has been made available publically at https://github.com/TimovNiedek/recsys-random-walk.

REFERENCES

[1] [n. d.]. Million Playlist Dataset. Official website, hosted at https://recsys-challenge.spotify.com. Accessed 2018-03-05.
[2] Gediminas Adomavicius and Alexander Tuzhilin. 2011. Context-aware recommender systems. In *Recommender systems handbook*. Springer, 217–253.
[3] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. RecSys Challenge 2018: Automatic Music Playlist Continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA.
[4] Maarten Clements, Arjen P. De Vries, and Marcel J. T. Reinders. 2010. The Task-dependent Effect of Tags and Ratings on Social Media Access. *ACM Trans. Inf. Syst.* 28, 4, Article 21 (Nov. 2010), 42 pages. https://doi.org/10.1145/1852102.1852107
[5] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. In *WWW*. 1775–1784.
[6] Dietmar Jannach, Iman Kamehkhosh, and Geoffray Bonnin. 2016. Biases in automated music playlist generation: A comparison of next-track recommending techniques. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*. ACM, 281–285.
[7] Fred Jelinek and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings, Workshop on Pattern Recognition in Practice*, Edzard S. Gelsema and Laveen N. Kanal (Eds.). North Holland, Amsterdam, 381–397.
[8] Jin Ha Lee, Bobby Bare, and Gary Meek. 2011. How Similar Is Too Similar?: Exploring Users' Perceptions of Similarity in Playlist Evaluation. In *ISMIR*. 109–114.
[9] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 462–470.
[10] Martin Pichl, Eva Zangerle, and Günther Specht. 2015. Towards a context-aware music recommendation approach: What is hidden in the playlist name?. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, 1360–1365.
[11] Martin Pichl, Eva Zangerle, and Günther Specht. 2017. Improving Context-Aware Music Recommender Systems: Beyond the Pre-filtering Approach. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, 201–208.
[12] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval* 7, 2 (2018), 95–116.
[13] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 8–pp.
[14] Timo van Niedek. 2018. *Random Walk with Restart for Automatic Playlist Continuation and Query-Specific Adaptations*. Master's thesis. Radboud University, Nijmegen.