Play It Again, Sam! Recommending Familiar Music in Fresh Ways

Giovanni Gabbolini
Insight Centre for Data Analytics, School of Computer Science & IT
University College Cork, Ireland
giovanni.gabbolini@insight-centre.org

Derek Bridge
Insight Centre for Data Analytics, School of Computer Science & IT
University College Cork, Ireland
d.bridge@cs.ucc.ie

ABSTRACT

In the music domain, repeated consumption is not uncommon. In this work, we explore how to recommend familiar music in fresh ways. Specifically, we design algorithms that can produce 'tours' through a small personal collection of songs. The tours are decorated with segues, which are textual connections between consecutive songs, chosen for their interestingness. We present three such algorithms, and we outline their strengths and weaknesses based on a comparative offline evaluation. This preliminary algorithmic work is a prelude to upcoming user-centric investigations.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

recommender systems, repeated consumption, segues

ACM Reference Format:
Giovanni Gabbolini and Derek Bridge. 2021. Play It Again, Sam! Recommending Familiar Music in Fresh Ways. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3460231.3478866

1 INTRODUCTION

Repeated consumption of music is a behavioural pattern that can be observed in users of music streaming services [2]. One way users may access familiar music is from a personal collection. For example, the streaming service Spotify offers its users the opportunity to create a virtual library of "saved" music.

Patterns of repeated consumption are sometimes used by music recommender systems to increase user engagement. For example, a recommender may deliberately surface songs that are familiar to a user, as well as novel songs [8]. The recommender may find interesting reasons to accompany the recommendation, for example "the last time you listened to this song was one year ago" [10].

In this work, we investigate one strategy to present users with their personal music collections in music streaming services. We strive to find an arrangement of the songs, and interesting textual links between those songs. We refer to the textual connections as *segues*. We refer to an arrangement of songs decorated by segues as a *tour*. An example of a short tour is in Figure 1. Our goal is to offer users a way to enjoy their music again, by presenting the songs in a fresh way, connected by interesting segues.

Here, we tackle the problem of finding tours through personal music collections in music streaming services. We formalise the problem and we propose three algorithms to solve the problem. Later, we offer an offline evaluation where we compare the characteristics of tours produced by those algorithms. Our offline evaluation is a preliminary analysis that does not yet take into account the user perspective. But, we have ethical approval for a first user trial that will build on the results of this paper. The source code supporting this study is freely available.[1]

2 RELATED WORK

Segues were first introduced in [1], and defined in [4] as "short texts that connect two items". Both [1] and [4] focus on the music domain and, in particular, the case where the items connected by the segues are songs. In [1], the authors developed a simple prototype able to find song-to-song segues for consecutive songs in an input playlist, to produce a list of segues to decorate the playlist. Our work in this paper is different from [1], as we do not consider the order of the songs to be fixed in advance. Instead, we strive to arrange the songs, and to produce a list of interesting segues to decorate this arrangement of the songs. In [4], the authors propose a domain-independent method to generate item-to-item segues from a knowledge graph. A distinguishing feature of their work is the introduction of a scoring function for segues, based on their interestingness. They evaluate their work in a user trial involving song-to-song segues. Our work in this paper builds on [4], as we describe in detail in Section 4.1.

Over the years, researchers have proposed many user interfaces to help users explore their personal music collections; see [5]. Our idea of using segues in this context is, to the best of our knowledge, a novel direction. Even so, there exist some user interfaces similar to ours. For example, Pohle et al. [7] arrange songs in a circle by solving a Travelling Salesman Problem, where the distances are determined by song-to-song similarity. Our work is similar to [7], because we also arrange songs one after the other. But our work is also different, because we include segues from one song to the next and the arrangement of the songs is based on the interestingness of the segues, not the similarity of the songs.

RecSys '21, September 27-October 1, 2021, Amsterdam, Netherlands

ACM ISBN 978-1-4503-8458-2/21/09.
https://doi.org/10.1145/3460231.3478866

[1] https://github.com/GiovanniGabbolini/play-it-again-sam

Figure 1: A tour in a personal collection of three songs. Circles are songs; rectangles are segues.

3 METHOD

3.1 Problem formulation

Let I be all or part of a person's music collection in a music streaming service. It is reasonable to assume I to be a 'small' set of songs, e.g. several order of magnitudes smaller than the whole catalogue of songs. Our goal is to find a tour of the songs in I, so as to obtain a solution of the kind shown in Figure 1.

More formally, given two songs $i, i' \in I$, let $segues(i, i')$ be the set of segues from i to i'. We assume that $segues(i, i')$ always contains at least one segue, as it is always possible to find a *null* segue [1]. Let L_j be the jth element in a list L, given that $j \in \{0, \ldots, |L| - 1\}$. Let $\mathcal{P}(I)$ be the permutations of set I. Then let $\mathcal{T}(I)$ be decorated permutations of I:

$$\mathcal{T}(I) = \{\langle O, S\rangle : S_j \in segues(O_{j-1}, O_j), \forall j \in \{1, \ldots, |I| - 1\}, \forall O \in \mathcal{P}(I)\} \quad (1)$$

In other words, $\mathcal{T}(I)$ is all the candidate solutions, and a given candidate solution $\langle O, S\rangle \in \mathcal{T}(I)$ comprises an ordering O of the items in I, and a corresponding sequence of segues S.

Problem 3.1 (Tour Finding Problem). *Given a set of songs I, find a solution $\langle O, S\rangle \in \mathcal{T}(I)$ that maximises some $utility(\langle O, S\rangle)$.*

In this paper, we define $utility(\langle O, S\rangle)$ in a simple way, based only on the interestingness scores for the segues, S. Let $score(s)$ be a real number ranging from zero to one. Then we define the utility of a solution $\langle O, S\rangle$ as the mean score of the segues:

$$utility(\langle O, S\rangle) = \frac{\sum_{s \in S} score(s)}{|I| - 1}$$

$utility(\langle O, S\rangle)$ is a real number ranging from zero to one.

More sophisticated definitions of $utility$ are, of course possible. For example, we might reward solutions in which similar songs are near each other in O, or where similar segues are more distant from each other in S. These are matters for future exploration with users. In this paper, we focus on obtaining insights into different algorithms for finding segues with high utility, rather than on the best definition of utility.

Notice that, even though our focus in this work is on repeated consumption of music, Problem 3.1 can be solved to find a tour of any set of songs, including ones that are new to the user. Also, notice that Problem 3.1 is NP-hard; intuitively it is related to the Travelling Salesman Problem. We make the analogy clearer in Section 3.2.3.

3.2 Algorithms

We introduce three algorithms for Problem 3.1: two are heuristic methods (Greedy and Hill-Climbing) and one is an exact algorithm (Optimal).

3.2.1 Greedy. The Greedy algorithm builds a solution iteratively, by choosing the next song to be the one with the segue of highest score. We give pseudo-code in Algorithm 1.

3.2.2 Hill-Climbing. Hill-climbing is a local search algorithm that starts from a random candidate solution, then iteratively replaces that candidate by a neighbouring candidate whose utility is highest; it stops if no neighbour would result in an improvement in utility. Many flavours of hill-climbing exist, e.g. see [9]. We adopt the version with two parameters: restarts and patience. So, we run the algorithm multiple times (the restarts) and, for a certain number of iterations (the patience), we tolerate replacement by neighbours even if none of them improves the utility.

In order to use hill-climbing for Problem 3.1, we have to decide how to define the neighbourhood of a candidate solution. In this paper, we define the neighbourhood of a solution (a tour) as all the solutions that can be obtained by swapping two consecutive songs at random. More formally, given a candidate solution $\langle O, S\rangle$ and a random $r \in \{2, \ldots, |I|\}$, the neighbourhood of the candidate induced by r is:

$$N(\langle O, S\rangle, r) = \{\langle O', S'\rangle \in \mathcal{T}(I) : O' = [O_1, \ldots, O_r, O_{r-1}, \ldots, O_{|I|}]\}$$

The members of this set of neighbours share the same new ordering of the items but they differ in their segues.

We give pseudo-code for Hill-Climbing in Algorithm 2.

3.2.3 Optimal. The Optimal algorithm finds an optimal solution t^* to Problem 3.1, i.e. a solution to Problem 3.1 with maximum $utility$.

Given a set of songs I, the algorithm builds a complete and weighted graph $\mathcal{G}(I)$. The nodes of $\mathcal{G}(I)$ are the songs I. $\mathcal{G}(I)$ is complete, so between every two songs there is an edge. $\mathcal{G}(I)$ is weighted, so every edge has a weight, equal to one minus the score of the highest-scoring segue between the two songs. More formally, the weight of the edge between two nodes (songs) $i, i' \in I$ is:

$$w(i, i') = 1 - max_{s \in segues(i,i')} score(s). \quad (2)$$

A Hamiltonian path $H = [i_1, \ldots i_{|I|}]$ in $\mathcal{G}(I)$ is a path in $\mathcal{G}(I)$ that visits every song exactly once. The set of all H in $\mathcal{G}(I)$ is equal to the permutations of I, and so it follows that $H \in \mathcal{P}(I)$. We define the weight W of H as:

$$W(H) = \sum_{j=1}^{|I|-1} w(H_j, H_{j+1}) \quad (3)$$

Intuitively, an H corresponds to a solution $t \in \mathcal{T}(I)$. Also intuitively, a Hamiltonian path with minimum weight H^* corresponds to the

Algorithm 1: GREEDY

$O \leftarrow$ empty list
$i \leftarrow$ random element from I
append i to O; remove i from I
while $|I| > 0$ **do**
  $i^* \leftarrow$ $arg\,max_{i' \in I}(\, max_{s \in segues(i,i')} score(s))$
  $s^* \leftarrow arg\,max_{s \in segues(i,i^*)} score(s)$
  append i^* to O; remove i^* from I
  append s^* to S
  $i \leftarrow i^*$
return $\langle O, S \rangle$

Algorithm 2: HILL-CLIMBING

$candidate\_solutions \leftarrow$ empty list
while $restarts > 0$ **do**
  $patience\_left \leftarrow patience$
  $current \leftarrow$ random element from $\mathcal{T}(I)$
  while *True* **do**
    $r \leftarrow$ random element from $\{2, \ldots, |I|\}$
    $neighbor \leftarrow arg\,max_{t \in N(current,r)} utility(t)$
    if $utility(neighbor) \geq utility(current)$ **then**
      $patience\_left \leftarrow patience$
      $current \leftarrow neighbor$
    else
      if $patience\_left > 0$ **then**
        $patience\_left \leftarrow patience\_left - 1$
        $current \leftarrow neighbor$
      else
        break
  append $current$ to $candidate\_solutions$
return $arg\,max_{t \in candidate\_solutions} utility(t)$

optimal solution t^*. More formally, it is possible to define a function f to map an H to a t, as follows:

$$f(H) = \langle H, S \rangle \;\; where$$
$$S_j = arg\,max_{s \in segues(H_j, H_{j+1})} score(s),\ \forall j \in \{1, \ldots, |I| - 1\}$$

$f(H)$ satisfies the membership conditions expressed in Equation 1, so $f(H) \in \mathcal{T}(I)$.

LEMMA 3.2. *There is an H such that $utility(f(H)) = utility(t^*)$.*

PROOF. Let $t^* = \langle O, S \rangle$. O is a permutation of I and so it is a valid Hamiltonian path H. By construction, $utility(f(H)) = utility(t^*)$. □

LEMMA 3.3. *H^* is such that $utility(f(H^*)) = utility(t^*)$.*

PROOF. By contradiction, we assume that $utility(f(H^*)) \neq utility(t^*)$. t^* is the optimal solution, so $utility(f(H^*)) < utility(t^*)$. Because of Lemma 3.2, there is an $\overline{H}$ such that $utility(f(\overline{H})) = utility(t^*)$. $\overline{H} \neq H^*$, as $utility(f(\overline{H})) = utility(t^*) \neq utility(f(H^*))$. Also, $W(H^*) \leq W(\overline{H})$, because H^* is the Hamiltonian path with minimum weight. From Equations 3 and 2, we get the following:

$$\sum_{j=1}^{|I|-1} 1 - max_{s^* \in segues(H_j^*, H_{j+1}^*)} score(s^*)$$
$$\leq \sum_{j=1}^{|I|-1} 1 - max_{\overline{s} \in segues(\overline{H}_j, \overline{H}_{j+1})} score(\overline{s})$$

$$\sum_{j=1}^{|I|-1} max_{s^* \in segues(H_j^*, H_{j+1}^*)} score(s^*)$$
$$\geq \sum_{j=1}^{|I|-1} max_{\overline{s} \in segues(\overline{H}_j, \overline{H}_{j+1})} score(\overline{s})$$

and so $utility(f(H^*)) \geq utility(f(\overline{H})) = utility(t^*)$, which contradicts the hypothesis. □

In conclusion, it is enough to compute $f(H^*)$ to obtain an optimal solution t^*. One way of finding H^* is by solving a Travelling Salesman Problem, or TSP. As suggested by [6], we can add a dummy node n to $\mathcal{G}(I)$ with zero-weighted edges to all the songs, then solve the TSP with start and end in n, and finally exclude n from the solution to obtain H^*. In the experiments that we run in this paper, we use the Concorde TSP solver.[2]

4 EXPERIMENTS

4.1 Implementation

The algorithms of Section 3 assume the existence of two functions: *segues* and *score*. We resort to the implementation of those two function proposed in [4]. In [4], the *score* function is based on the interestingness of the segues. Their *segues* function can find segues of two kinds: informative (based on a knowledge graph) and funny (based on simple word-play). In the work reported in this paper, we restrict to informative segues. In a user trial, [4] finds that the interestingness of informative segues is correlated with human perceptions of segue quality.

4.2 Dataset

We build a dataset from the Spotify Million Playlists Dataset (MPD). [3]. The MPD dataset contains user-created playlists. A user-created playlist is typically a subset of a user's personal music collection, which is what our algorithms take as input. Of course, we ignore the ordering of the playlist, treating it as a set rather than a list, because our algorithms impose a fresh ordering, as well as decorating with segues.

[2] https://github.com/jingw2/pyconcorde

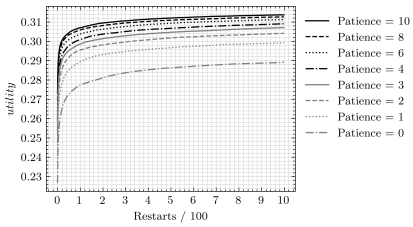

Figure 2: Average *utility* of Hill-Climbing, as a function of the restarts and of the patience.

We limit ourselves to playlists of maximum 50 songs. We believe longer inputs to be unlikely in practice, as it would lead to a listening time of more than three hours, assuming every song to last four minutes. We sample the MPD with stratified random sampling by playlist length: we sample at random 20 playlists of length 50, 20 of length 49, and so on, down to 20 playlists of length five, as five is the minimum length of MPD playlists. In fact, we apply this procedure twice, with two different random seeds, to obtain a *main* dataset and a *side* dataset.

4.3 Parameter tuning

Hill-Climbing has two parameters, as described in Section 3.2.2: the patience and the restarts. We set the parameters by choosing the configuration that maximises the average *utility* of solutions. We run Hill-Climbing on the *side* dataset of Section 4.2, with various parameter configurations, and we measure the average *utility* of the solutions. We report the results in Figure 2. We notice that the average *utility* grows with patience and restarts, until it saturates. We choose the parameter values to be those before saturation, and so we set the patience to 10 and the restarts to 40.

4.4 Performance

We benchmark the algorithms by monitoring the *score* of the segues in the tours they produce using the *main* dataset. We construct four curves showing the mean of: average *score*; standard deviation; maximum; and minimum *score* of segues in the solutions, as a function of the input size. The average *score* of the segues in a solution is equivalent to the *utility* of the solution. For presentation purposes, we do a least square fitting and plot the fitted curves in Figure 3.[3]

Greedy, by construction, produces segues of high *score* at the beginning of a solution, and of low *score* at the end of a solution. That is, the segues in the solutions found by Greedy exhibit falling average *score*, with highest values of maximum *score*, and lowest values of minimum *score*, which result in the highest standard deviation. The other two algorithms, Hill-Climbing and Optimal, do not share the characteristic of falling average *score*, and produce solutions that are different from those produced by Greedy. The solutions found by Optimal are characterised by highest average *score*, and high values of maximum and minimum *score*, which contribute to the lowest values of standard deviation. Figure 3

[3]We use a polynomial of degree three, as we find empirically that it produces a good fit.

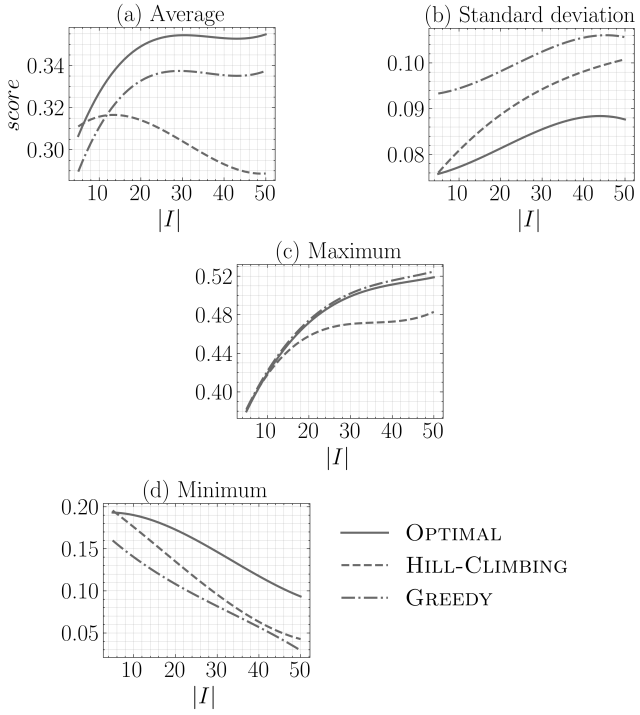

Figure 3: Performance of the algorithms, as a function of the input size.

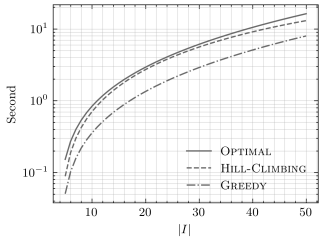

Figure 4: Granular runtimes of the algorithms, as a function of the input size.

provide some indication that Hill-Climbing and Optimal produce, in general, the same solutions for small inputs. In the case of Hill-Climbing, as the input size grows, the average *score* falls, the maximum stops increasing, and the minimum falls, causing an increase in standard deviation. As we see, Hill-Climbing struggles for large inputs, but it is likely to find the optimal solution for small inputs. On the other hand, Greedy behaves similarly for small and large inputs, but it is unlikely to find the optimal solution for both small and large inputs.

4.5 Runtime

We compare the times each algorithm requires from taking an input to producing an output. More specifically, we carry out what we will call a granular analysis. By this, we mean that we consider the time required by calls to the functions *segues* and *score*. All

the algorithms call these functions, but they call them a different number of times. There are other parts of each algorithm which are not shared, and that are written in different programming languages, e.g. the optimal TSP solver is in C, while the hill-climber is in Python. We found that the granular runtimes, computed as above, differ from the full runtimes by only a small extent, and so we believe that the granular runtimes give us a fair comparison.

We run the algorithms on the *main* dataset of Section 4.2. We construct a curve featuring the granular runtimes as a function of the input size. For presentation purposes, we do a least square fitting, as in Section 4.4. We report the fitted curve in Figure 4. We find that Greedy is the fastest, while Optimal is the slowest. Hill-Climbing is slightly faster than Optimal, and much slower than Greedy. The runtime of Hill-Climbing might decrease if we decrease the values of its parameters, at the cost of lower performances, as we discussed in Section 4.4.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we introduce a novel strategy to present users with their personal music collections, based on what we call tours. A tour is an arrangement of the songs, with segues between consecutive songs. We introduce three algorithms that can find tours: two heuristics methods (Greedy and Hill-Climbing) and one exact algorithm (Optimal). The results of our experiments highlight that Greedy produces tours that are substantially different from those found by Hill-Climbing and Optimal. Tours found by Greedy have segues with falling *scores* but highest maximum *scores*. Hill-Climbing and Optimal find similar tours to each other for small input sizes, and do not have falling *scores*.

Future work will involve user-centric evaluation; we have ethical approval for a first user trial that will build on the results of this paper. We plan to gather human opinions on the different tours produced by the algorithms, with a focus on the practical implication of the results found in this paper. Also, we plan to gather human opinions on possible improvements to the tours, and on the potential impact of tours on repeated consumption of music.

ACKNOWLEDGMENTS

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 12/RC/2289-P2 which is co-funded under the European Regional Development Fund. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] Morteza Behrooz, Sarah Mennicken, Jennifer Thom, Rohit Kumar, and Henriette Cramer. 2019. Augmenting Music Listening Experiences on Voice Assistants. In *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands, November 4-8, 2019*, Arthur Flexer, Geoffroy Peeters, Julián Urbano, and Anja Volk (Eds.). 303–310. http://archives.ismir.net/ismir2019/paper/000035.pdf

[2] Austin R. Benson, Ravi Kumar, and Andrew Tomkins. 2016. Modeling User Consumption Sequences. In *Proceedings of the 25th International Conference on World Wide Web* (Montréal, Québec, Canada) *(WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 519–529. https://doi.org/10.1145/2872427.2883024

[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* (Vancouver, British Columbia, Canada) *(RecSys '18)*. Association for Computing Machinery, New York, NY, USA, 527–528. https://doi.org/10.1145/3240323.3240342

[4] Giovanni Gabbolini and Derek Bridge. 2021. Generating Interesting Song-to-Song Segues With Dave *(UMAP '21)*. Association for Computing Machinery, New York, NY, USA, 98–107. https://doi.org/10.1145/3450613.3456819

[5] Peter Knees, Markus Schedl, and Masataka Goto. 2020. Intelligent User Interfaces for Music Discovery. *Transactions of the International Society for Music Information Retrieval* 3(1) (2020), 165–179.

[6] J. K. Lenstra and A. H. G. Rinnooy Kan. 1975. Some Simple Applications of the Travelling Salesman Problem. *Journal of the Operational Research Society* 26, 4 (1975), 717–733. https://doi.org/10.1057/jors.1975.151

[7] T. Pohle, P. Knees, M. Schedl, E. Pampalk, and G. Widmer. 2007. "Reinventing the Wheel": A Novel Approach to Music Player Interfaces. *IEEE Transactions on Multimedia* 9, 3 (2007), 567–575. https://doi.org/10.1109/TMM.2006.887991

[8] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten de Rijke. 2019. RepeatNet: A Repeat Aware Neural Recommendation Machine for Session-Based Recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 4806–4813. https://doi.org/10.1609/aaai.v33i01.33014806

[9] Stuart Russell and Peter Norvig. 2010. Beyond Classical Search. In *Artificial Intelligence: A Modern Approach 3rd Ed.* Prentice Hall, Upper Saddle River, NJ, 120–160.

[10] Kosetsu Tsukuda and Masataka Goto. 2020. Explainable Recommendation for Repeat Consumption. In *Fourteenth ACM Conference on Recommender Systems* (Virtual Event, Brazil) *(RecSys '20)*. Association for Computing Machinery, New York, NY, USA, 462–467. https://doi.org/10.1145/3383313.3412230