Journal of Turkish

Operations Management

Mathematical modelling and a greedy heuristic for harmonic mixing and popularity-based playlist generation problem

Zülkar Karakaya[1], Zeliha Ergül Aydın[2*]

[1] Department of Industrial Engineering, Eskisehir Technical University, Eskişehir
e-mail: zulkarkarakaya1@gmail.com, ORCID No: http://orcid.org/0009-0008-2636-0684
[2] Department of Industrial Engineering, Eskisehir Technical University, Eskişehir
e-mail:zergul@eskisehir.edu.tr, ORCID No: http://orcid.org/0000-0002-7108-8930
*Corresponding Author

Article Info

Article History:
Received: 08.11.2023
Revised: 16.09.2024
Accepted: 22.10.2024

Keywords

Harmonic mixing,
Playlist generation,
Greedy algorithm

Abstract

The rise of the electronic music industry has led to a need for creative playlist-generation methods, particularly for DJs aiming to deliver seamless and harmonically enhanced performances. Harmonic mixing, a crucial process of DJing, involves synchronizing and aligning songs based on musical harmony, making the mix sound soft and clear. In harmonic mixing, the DJ has to select songs from the extensive music archive, considering notes, tempo, length, and popularity of the songs. However, manually generating playlists that adhere to harmonic mixing principles can be time-consuming. This paper introduces a mixed-integer mathematical model and a novel greedy heuristic to automate playlist generation, considering factors like popularity, tempo, and harmonic mixing rules. We compare the novel greedy heuristic's performance to the mathematical model on 16 test problems created with Spotify's API, incorporating real-world data on song characteristics. The results show that the heuristic method generates playlists at least seven times faster and has an average gap of 13.84% with the mathematical model.

1. Introduction

The electronic music industry has grown significantly in recent years, driven by the increasing number of music fans, DJs, and live events. One of the main tasks of DJs today is to generate seamless playlists that captivate the audience and elevate the overall experience. Generating such playlists, however, is a complex and laborious task considering notes, tempo, length, and song popularity.

Harmonic mixing has become vital for DJs seeking to deliver polished performances. DJs apply the method of harmonic mixing to beat-synchronize and harmonically align two or more songs (Gebhardt, Davie, and Seeber, 2016). Despite its importance, existing literature lacks comprehensive studies combining harmonic mixing principles with other critical factors such as song popularity and tempo transitions. The importance of harmonic mixing cannot be underestimated because it contributes significantly to the integrity and clarity of a DJ's performance. However, given the many factors to consider, manually editing a playlist that adheres to harmonic mixing principles can be time-consuming and challenging. Accordingly, our research aims to help DJs by enabling them to quickly create playlists automatically while ensuring harmony in notes, tempo, popularity, and playlist duration. This study is unique in its combination of harmonic mixing principles with other key factors such as song popularity, tempo transitions, and playlist duration an approach not previously explored in automatic playlist generation studies.

This paper presents a mixed-integer mathematical model and a novel greedy heuristic designed to generate playlists that adhere to harmonic mixing principles. Our mathematical model aims to maximize the total popularity of the songs within the playlist, considering playlist duration and tempo transitions. Given the computational complexity of the model, we develop a greedy heuristic algorithm. The greedy heuristic algorithm builds playlists

by strategically adding songs based on popularity, suitability for harmonic mixing, playlist duration, and tempo transitions, which start by selecting a random first song. To validate the mathematical model and greedy heuristic, we apply them to a diverse set of 16 test problems of varying sizes, drawing upon data obtained through Spotify's API, which includes information on song popularity, key, notes, tempo, and duration.
In the subsequent sections of this paper, we examine the related works, give the problem statement, mathematical model, and greedy heuristic algorithm in detail, present the results from our comprehensive computational testing, and analyze these results.

2. Related works

Dias et al. (2017) categorized the playlist generation problems under three headings; manual generation, automatic generation and recommendation, and assisted playlist creation. This study considers automated methods that create playlists nearly entirely without human interaction. Automatic playlist generation has become one of the main fields in music information retrieval (Shuhendler and Rabin, 2024). Mainly, automatic playlist generation approaches are similarity-based algorithms, collaborative filtering, frequent pattern mining, statistical models, case-based reasoning, discrete optimization, and hybrids (Bonnin and Jannach, 2014). This section focuses on discrete optimization techniques for automatic playlist generation, as the proposed methods in this study are under this category. In discrete optimization, the goal is to create a sequence of songs from a catalogue that meets predefined constraints, reflecting the desired rules, while maximizing a utility function (Gabbolini and Bridge, 2024). Automatic playlist generation is NP-hard (Pauws et al., 2008), hence existing studies about discrete optimization performed metaheuristic or heuristic algorithms to solve large-size problems.

The Traveling Salesman Problem (TSP) can be used to translate the challenge of creating a playlist that satisfies specific constraints into a problem of determining the optimal route (Hartono and Yoshitake, 2013). Pohle et al. (2005) presented an approach to generate playlists for mobile music players that prioritize similarity between consecutive songs, using a TSP algorithm with timbral similarities as distances. They evaluated the fitness of four different TSP algorithms based on runtime, route length, and genre distribution entropy. Pohle et al. (2007), who again address the problem as TSP, presented a new interface for music players that uses a combination of audio signal analysis and web-based artist profile comparison to create intelligently structured playlists with minimal distance between songs. On the other Mocholi et al. (2012) handled playlist generation problem as an Orienteering Problem and proposed multicriteria ant colony algorithm for a solution. Why they used OP instead of TSP is that they considered playlists created with some songs selected from music archives. Hsu and Lai (2014) applied a tabu search algorithm to a generate playlist automatically for users based on user listening history. All these studies in discrete optimization generated personalized playlists based on the user's music preference. Discrete optimization-based research on automatic playlist generation appears to have stopped after 2014 (Gabbolini and Bridge, 2024). This could be because of the growing popularity of machine learning and artificial intelligence techniques. However, discrete optimization approaches still hold significant promise for developing customized and effective automatic playlist creation, especially for DJs with specific needs, such as harmonic mixing requirements.

There are a limited number of studies on automatic playlist generation for DJs. Kahanda and Kanewala (2007), combined similarity-based and constraint-based techniques to generate playlists automatically for radio stations. They ensure two songs of the same album/artist are not included in a playlist, the genre of two consecutive songs should be different, and the tempos of consecutive songs should be very close with Improved Adaptive Search Algorithm. Besides, they use content-based similarity functions to measure the similarity of songs. To our knowledge, none of these studies have addressed harmonic mixing rules for playlist generation problems. To fill this gap, we formulate the playlist generation problem with harmonic mixing rules as a mixed-integer mathematical model and propose a novel greedy heuristic for the solution.

3. Problem statement and mathematical model

DJs generating playlists for an event should plan carefully and create an engaging musical experience for the audience of an event. When creating a playlist, DJs pick songs from their music archives that suit the event concept and the audience. Some musical rules apply when selecting and sorting songs.

Generally, people would rather listen to familiar songs than unfamiliar ones. One of the factors of a good playlist is the audience's familiarity with the songs in the playlist (Fields et al., 2010). Therefore, including popular songs in the playlist may attract the audience's attention.

Most DJs apply the rules of harmonic mixing when preparing the playlist, which they will use in their performances so that the transition between songs played back to back is smooth. The most crucial point of the harmonic mixing method is to mix by considering a mathematical set of rules when mixing songs. This way, the total harmony of the songs is ensured. Harmonic mixing is based on selecting notes created by the notation method called Camelot by paying attention to a mathematical rule. In Camelot's notes, 24 notes start from 1A and 1B to 12A and 12B. Whatever the note of the song playing at that moment is, the note of the next song must either have the same note as the song playing or have a note adjacent to the note of the song playing on the Camelot wheel shown in Figure 1. Figure 1 also shows the adjacent (neighbors) notes of 8A, which are 9A, 7A, and 8B. According to the harmonic mixing rule, a song with 8A, 7A, 9A, or 8B can be played after a song with the note 8A.

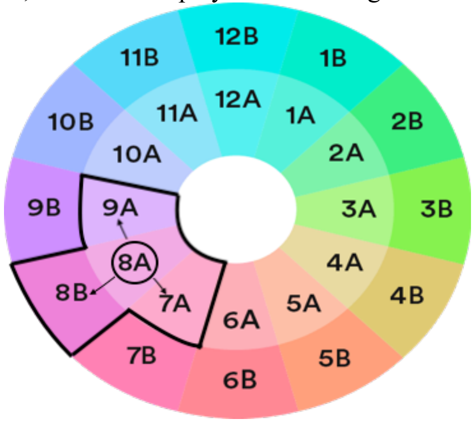

Figure 1. Camelot Wheel

The flow of the playlist is an aesthetic phenomenon; so, a DJ may want the tempo to stay relatively close between the consecutive songs in addition to consecutive songs to be acoustically similar (Bittner et al., 2017). The acoustically similarity can be achieved by ensuring that the bpm difference between the tempo of two consecutive songs in this playlist does not exceed a specific value.

DJs generate their playlists per all these rules manually by spending hours or by paying significant fees through paid programs. To solve this problem, first of all, a mixed integer mathematical model is presented in this study. Then, a greedy heuristic algorithm is proposed to solve large problems in a reasonable time. The definition of the sets, parameters, and decision variables of the mathematical model is as follows:

Sets:

I, J, K: The set of songs which in the music archive {1,…, n} 1:dummy starting song n: dummy last song

Parameters:

p_i : The popularity score of song i

t_i: The tempo of song i

s_i: The duration of song i

d_{ij}: $\begin{cases} 1, if\ song\ i\ and\ song\ j\ are\ neighbors\ on\ camelot\ wheel \\ 0, otherwise \end{cases}$

b: The maximum tempo changes between two consecutive songs in the playlist

c: The playlist's duration in seconds

n: number of songs in the music archive

Decision variables:

$$x_{ij} = \begin{cases} 1, \text{if song j plays after song i in the playlist} \\ 0, \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1, \text{if the song i plays in the playlist} \\ 0, \text{otherwise} \end{cases}$$

$$\text{Max}\, z = \sum_{i=1}^{n} p_i y_i \tag{1}$$

s.t

$$\sum_{i=1}^{n} s_i y_i \le c \tag{2}$$

$$\sum_{\substack{j=1 \\ i\neq j}}^{n} x_{ij} = y_i \quad \forall i \tag{3}$$

$$x_{ij} \le d_{ij} \quad \forall i,j \in I \tag{4}$$

$$x_{ij}\left|t_i - t_j\right| \le b \quad \forall i,j \tag{5}$$

$$\sum_{j=2}^{n} x_{1j} = 1 \tag{6}$$

$$\sum_{i=1}^{n-1} x_{in} = 1 \tag{7}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} \le 1 \quad \forall j \tag{8}$$

$$\sum_{\substack{j=1 \\ i\neq j}}^{n} x_{ij} \le 1 \quad \forall i \tag{9}$$

$$\sum_{\substack{i=1, \\ i\neq j}}^{n-1} x_{ij} - \sum_{\substack{k=2 \\ j\neq k}}^{n} x_{jk} = 0 \quad \forall j \tag{10}$$

$$u_i - u_j + 1 \le (n-1)\left(1 - x_{ij}\right) \quad \forall i, \forall j\, , i \ne j \tag{11}$$

$$0 \le u_i \le n-1 \quad \forall i \tag{12}$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \tag{13}$$

$$y_i \in \{0,1\} \quad \forall i \tag{14}$$

$$0 \le u_i \quad \forall i \tag{15}$$

The objective function (1) maximizes the popularity of the playlist. The capacity constraint (2) ensures that the total duration of the songs in the playlist does not exceed the total duration of the playlist. If a song plays before another song, it will be assigned to the playlist in (3). (4) assures that songs can only be played back to back on the playlist if they are neighbors on Camelot wheel. This constraint ensures that the playlist fits the harmonic mixing rules. (5) checks that the difference between the tempos of consecutive songs in the playlist does not exceed the determined maximum tempo change limit. (6) and (7) make sure that the playlist starts and ends at the dummy starting and closing songs. (8) and (9) allow at most one song to be played before and after each song. The flow constraint of the model is given in (10). (11) represent Miller-Tucker-Zemlin subtour elimination constraints. In this problem, this subtour elimination constraint prevents the generation of subplaylists. (12) to (15) define decision variables, respectively.

4. Proposed method: A greedy heuristic

The proposed method for automatic playlist creation utilizes a greedy heuristic algorithm designed for efficiency and effectiveness in generating playlists based on harmonic mixing principles. The algorithm aims to maximize the playlist's popularity, considering constraints such as tempo differences, harmonic mixing rules, and the playlist's desired duration. The algorithm starts with a randomly selected song and builds a playlist iteratively by greedily selecting the next most popular song that ensures the constraints. The greedy heuristic algorithm is demonstrated in Table 1.

Table 1. Pseudocode of the greedy heuristic algorithm

Input: *I* Music archive, *b* the maximum tempo changes between two consecutive songs in the playlist, *c* the playlist's duration in seconds
Output: *P* Playlist

```
maxPopularite=0
For i=1 to populationSize:
  Initialize I, candidatePlaylist
  Select a song randomly from I as a currentSong
  Add currentSong to candidatePlaylist
  currentDuration = currentSong’s duration
  currentPopularite= currentSong’s popularity
  while currentDuration <= c:
    feasibleSongList={}
      For j in I:
       if song j neighbors of current song in Camelot and tempo(song j)-tempo(currentSong)<=b
          Add song j to feasibleSongList
      if length (feasibleSongList)==0:
        Break
      else:
        Assign the song with maximum popularity in the feasibleSongList as the currentSong
        currentDuration+= current_song’s duration
        currentPopulerite+= current_song’s popularity
        Add current_song to candidatePlaylist
        Remove current_song from I
        if maxPopularite< currentPopulerite:
          maxPopularite= currentPopulerite
          P= candidatePlaylist
```

5. Computational experiments and discussions

5.1 Generation of test problems

One of the world's most popular music streaming platforms, Spotify allows users to listen to songs on smart devices from prominent international record companies without purchasing songs legally. Besides, developers can retrieve content metadata with Spotify Web API (2023) for free. We also use the Spotify Web API to generate test problems for this study.

With Spotify Web API, we accessed 16 public electronic music archives and the songs' characteristics in these archives: popularity, tempo, key, mode, and duration. These music archive lists contain different numbers of songs from 20 songs to 1077 songs. A song's popularity score ranges from 0 to 100, with 100 being the most popular. The number of plays the song has received and how recently those plays have occurred are used to calculate the popularity score. The key value representing the pitch class takes a value between 0 and 11, and a mode value of 1 or 0 represents major or minor. Harmonic mixing is usually based on a notation called Camelot. For this reason, we converted the key value and mode values to notes of Camelot notation. Finally, duration is the song length in milliseconds. We used Python 3.6 to create the test problems.

5.2 Parameters

While solving test problems in this study, we set the maximum value of the tempo difference of two consecutive songs as 10 bpm and the playlist's duration in seconds as 3600300 ms. Also, for the greedy heuristic, we take the initial population size as 25% of the number of songs in the archive included in each test problem. Moreover, we run all experiments on a computer with a macOS Mojave, Intel Core i7 at 2.5GHz, 4 cores, and 16.00 GB RAM.

5.3 Computational results and discussions

To find out how long they can wait to see the results of a program that automatically generates a playlist based on harmonic mixing, we reached out to DJs. According to the responses we received, the working time of the model was limited to 3 hours. In addition, we employed Gurobi solver with Python to solve the proposed mathematical model. The results obtained from solving the test problems with the mathematical model are given in Table 2. In the given 3-hour solution time, the mathematical model reached the best solution for 11 of the 16 test problems. However, for the remaining 5 test problems, the best solution found within the given time was reported.

Table 2. Mathematical model results

Test Problem	Number of songs	Model status	Run time (sec)	Total popularity of the playlist
Test 1	20	optimal	0.470	401
Test 2	30	optimal	0.598	492
Test 3	40	optimal	1.061	492
Test 4	50	optimal	1.695	492
Test 5	60	optimal	7.945	503
Test 6	92	optimal	8.107	934
Test 7	101	best solution	10800.048	1117
Test 8	156	best solution	10800.095	537
Test 9	150	best solution	10800.104	1117
Test 10	200	optimal	1983.153	1021
Test 11	251	optimal	675.280	1687
Test 12	387	best solution	10800.357	1547
Test 13	788	optimal	296.008	1134
Test 14	860	optimal	103.828	1437
Test 15	1000	best solution	10804.308	1478
Test 16	1077	optimal	307.932	1399

As an example, Table 3 shows the playlist obtained with the mathematical model was created from an archive of 40 songs called Test 3.

Table 3. Playlist generated by the mathematical model for test 3

Song Name	Note	Tempo	Popularity	Duration
Kaizoku	6B	145	42	7 min 07 sec
Deine Angst	7B	144	52	5 min 21 sec
Full of Fire	8B	146	39	5 min 24 sec
Weltschmerz	8A	140	53	6 min 24 sec
Adrenaline	8B	134	49	6 min 51 sec
Still Raving	9B	135	45	5 min 57 sec

300000003	10B	133	52	5 min 45 sec
Born In 1968	10A	135	42	5 min 7 sec
Rave Harder Techno Bass	9A	136	69	6 min
Sparkling System	9A	136	49	5 min 58 sec
			Total: 492	Total: 59 min 54 sec

The greedy heuristic algorithm with random-start generates playlists with ¼ of the number of songs in the archive and selects which has the most total popularity. Because the first song in each playlist is randomly selected, the algorithm does not produce the same result every time. For this reason, the greedy heuristic algorithm was run ten times for each test problem to get more reliable results. The average results for each test are shown in Table 4. Table 4 demonstrates that the solution of the heuristic algorithm is below the total popularity values reached by the mathematical model. However, it produces solutions in a shorter time than the mathematical model.

Table 4. Greedy heuristic results

Test Problem	Number of songs	Average run time (sn)	Average total popularity
Test 1	20	0.0003	340.4
Test 2	30	0.0010	447.3
Test 3	40	0.0014	450.2
Test 4	50	0.0025	447.1
Test 5	60	0.0039	468.5
Test 6	92	0.0108	695.5
Test 7	101	0.0103	911.8
Test 8	156	0.0710	524.1
Test 9	150	0.0499	866.4
Test 10	200	0.0811	796.5
Test 11	251	0.3061	1343.9
Test 12	387	1.5459	1273.2
Test 13	788	14.8403	1050.8
Test 14	860	3.1289	1247.6
Test 15	1000	36.5382	1259.3
Test 16	1077	42.0617	1276.1

Run time analysis of the methods is given in Table 5 in detail. In the last column of the table, the greedy heuristic algorithm is given how many times faster the mathematical model is. According to run time, it is clear that the greedy heuristic algorithm surpasses the mathematical model. In the large-scale test problems 13 to 16, where the heuristic algorithm runs slowest, it has been observed to be approximately 14, 3, 36, and 42 times faster than the mathematical model, respectively. This result shows that the heuristic algorithm can be more useful in large-scale real-life problems.

Table 5. Run times

Test Problem	Run time of the mathematical model (sec)	Run time of the greedy heuristic (sec)	Relative Speed
Test 1	0.470	0.0003	1566.6667
Test 2	0.598	0.0010	598.0000
Test 3	1.061	0.0014	757.8571
Test 4	1.695	0.0025	678.0000
Test 5	7.945	0.0039	2037.1795
Test 6	8.107	0.0108	750.6481

Test 7	10800.048	0.0103	1048548.3495
Test 8	10800.095	0.0710	152114.0141
Test 9	10800.104	0.0499	216434.9499
Test 10	1983.153	0.0811	24453.1813
Test 11	675.280	0.3061	2206.0764
Test 12	10800.357	1.5459	6986.4526
Test 13	296.008	14.8403	19.9462
Test 14	103.828	3.1289	33.1835
Test 15	10804.308	36.5382	295.6990
Test 16	307.932	42.0617	7.3210

In Figure 2, the results of the mathematical model and the greedy heuristic algorithm are compared regarding the total popularity of playlists. From Figure 2, it can be seen that the greedy heuristic approach produces results that are comparable to those of the mathematical model when the playlists. In addition, Figure 3 shows the gap between the solutions obtained by the mathematical model and the greedy heuristic. This gap is calculated by (16). According to this comparison, the maximum gap between the greedy heuristic algorithm and the mathematical model in the Test 6 problem is 25.4%. This maximum gap is acceptable, considering the speed of the greedy heuristic algorithm. The test problem in which the heuristic algorithm comes closest to the result of the mathematical model is the Test 8 problem with a gap of 2.4%. This result emphasizes the capacity of the heuristic methodology to provide practical and effective solutions. Besides, it is seen that the greedy heuristic algorithm approaches the mathematical model with an average gap of 13.84% in all test problems. In particular, the gap value in large test problems 13, 14, and 16, in which the mathematical model reaches the optimal solution, is lower than the average openness. This finding indicates that the heuristic algorithm can find effective solutions for large-scale real-life problems in a short time.

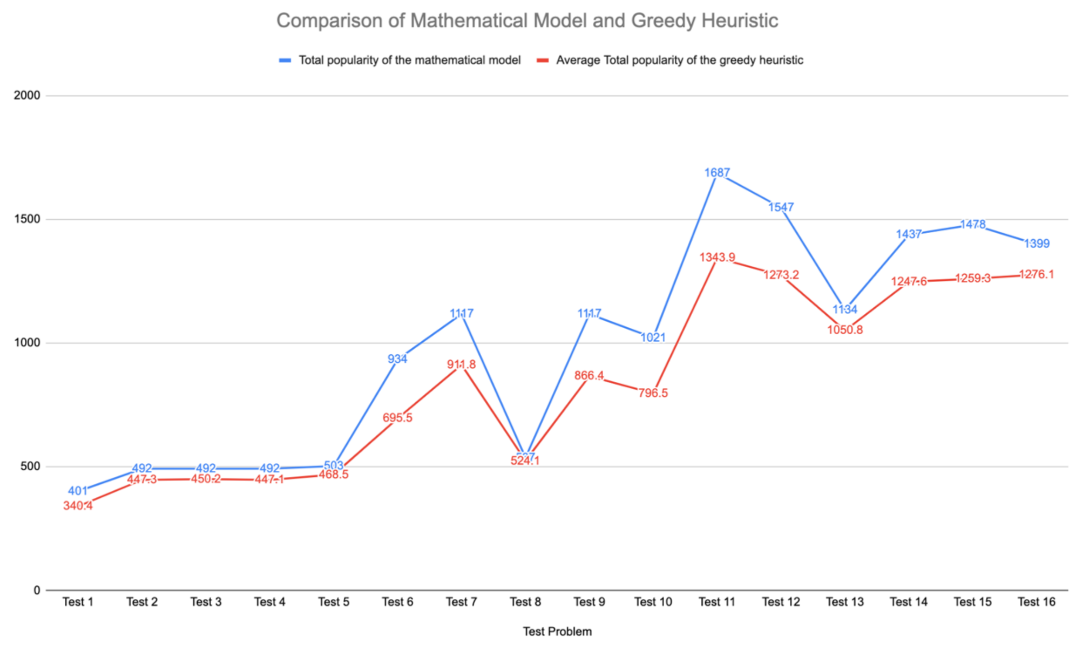

Figure 2. Comparison of mathematical model and greedy heuristic algorithm

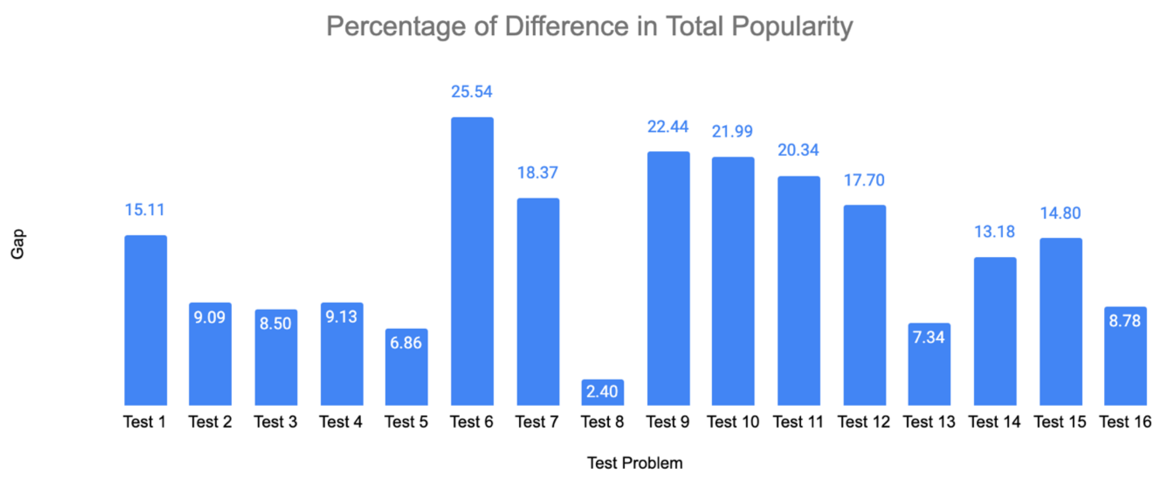

Figure 3. Gaps between mathematical model and greedy heuristic algorithm

$$\text{Gap} = \frac{\text{Mathematical Model Solution} - \text{The greedy heuristic solution}}{\text{Mathematical Model Solution}} . 100 \tag{16}$$

The greedy heuristic's primary goal is to solve the problem much more quickly than a mathematical model, which takes a significantly longer. In addition, the mathematical model is solved using the license-required Gurobi solver. The algorithm has a cost advantage because the heuristic does not require a Gurobi license, which is quite expensive. One can reach datasets, the codes of the mathematical model, and the heuristic algorithm used in this study on the https://drive.google.com/drive/folders/1qM3cLsHzzDx6U3IbjHVcYuXUo9sI_PN9?usp=sharing.

6. Conclusion and future research

This paper presented a new mathematical model for harmonic mixing and a popularity-based playlist generation problem. This model generated a playlist that maximized the total popularity, assured the harmonic mixing rules, and limited the tempo difference between songs played consecutively. The proposed model was evaluated on 16 test problems created using real-music archives via Spotify API. While the model reached the optimal solution in some of these test problems, it reached the best possible solution in others. Based on this, we proposed a greedy heuristic. The heuristic method generated playlists at least 7 times faster than the mathematical model in test problems. In addition to its short execution time, the heuristic had an average gap of 13.84% across all test problems compared to the mathematical model. The heuristic algorithm provided promising solutions in a short time. Future research could explore metaheuristic algorithms, such as genetic algorithms, to solve this problem more efficiently. Additionally, a decision support system could be developed to help DJs generate playlists easily.

Contribution of authors

Zülkar Karakaya was responsible for data collection, data curation, writing, software development, validation, and visualization. Zeliha Ergül Aydın was responsible for methodology, software development, validation, data curation, supervision, and writing.

Acknowledgement

This study was supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK) 2209/A Program and Eskisehir Technical University Scientific Research Projects Commission under grant no: 22LÖP394.

Conflicts of interest

There is no conflict of interest in this study.

References

Bittner, R. M., Gu, M., Hernandez, G., Humphrey, E. J., Jehan, T., McCurry, H., & Montecchio, N. (2017, October). Automatic Playlist Sequencing and Transitions. In ISMIR (pp. 442-448).

Bonnin, G., & Jannach, D. (2014). Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2), 1-35. https://dl.acm.org/doi/10.1145/2652481

Dias, R., Gonçalves, D., & Fonseca, M. J. (2017). From manual to assisted playlist creation: a survey. Multimedia Tools and Applications, 76, 14375-14403. https://doi.org/10.1007/s11042-016-3836-x

Fields, B., Lamere, P., & Hornby, N. (2010, August). Finding a path through the juke box: The playlist tutorial. In 11th International Society for Music Information Retrieval Conference (ISMIR).

Gabbolini, G., & Bridge, D. (2024). Surveying More Than Two Decades of Music Information Retrieval Research on Playlists. *ACM Transactions on Intelligent Systems and Technology*. https://doi.org/10.1145/3688398

Gebhardt, R., Davies, M., & Seeber, B. (2016). Psychoacoustic Approaches for Harmonic Music Mixing. *Applied Sciences*, 6(5), 123. https://doi.org/10.3390/app6050123

Hartono, P., & Yoshitake, R. (2013). Automatic playlist generation from self-organizing music map. *Journal of Signal Processing*, 17(1), 11-19. https://doi.org/10.2299/jsp.17.11

Hsu, J. L., & Lai, Y. C. (2014). Automatic playlist generation by applying tabu search. *International Journal of Machine Learning and Cybernetics*, 5, 553-568. https://doi.org/10.1007/s13042-013-0151-y

Kahanda, I., & Kanewala, U. (2007) PlayGen: A HYBRID PLAYLIST GENERATOR, *in Annual Technical Conference 2007 of IET-YMS*

Mocholi, J. A., Martinez, V., Jaen, J., & Catala, A. (2012). A multicriteria ant colony algorithm for generating music playlists. *Expert Systems with Applications*, 39(3), 2270-2278. https://doi.org/10.1016/j.eswa.2011.07.131

Pauws, S., Verhaegh, W., & Vossen, M. (2008). Music playlist generation by adapted simulated annealing. *Information Sciences*, 178(3), 647-662. https://doi.org/10.1016/j.ins.2007.08.019

Pohle, T., Pampalk, E., & Widmer, G. (2005, September). Generating similarity-based playlists using traveling salesman algorithms. *In Proceedings of the 8th International Conference on Digital Audio Effects (DAFx-05)* (pp. 220-225).

Pohle, T., Knees, P., Schedl, M., Pampalk, E., & Widmer, G. (2007). "Reinventing the wheel": a novel approach to music player interfaces. *IEEE Transactions on Multimedia*, 9(3), 567-575. https://doi.org/10.1109/TMM.2006.887991

Shuhendler, R., & Rabin, N. (2024). Dynamic artist-based embeddings with application to playlist generation. Engineering Applications of Artificial Intelligence, 129, 107604. https://doi.org/10.1016/j.engappai.2023.107604

SpotifyWebAPI, Spotify for developers, 2023. Available a https://developer.spotify.com/documentation/web-api