GENERATING SIMILARITY-BASED PLAYLISTS USING TRAVELING SALESMAN ALGORITHMS

Tim Pohle[1], *Elias Pampalk*[1], *Gerhard Widmer*[1,2]

[1]Austrian Research Institute for Artificial Intelligence
Schottengasse 6/6, A-1010 Vienna, Austria
[2]Department of Computational Perception
Johannes Kepler University (JKU) Linz
Altenberger Str. 69, A-4040 Linz, Austria
tim@ofai.at, elias@ofai.at, gerhard.widmer@jku.at

ABSTRACT

When using a mobile music player en-route, usually only little attention can be paid to its handling. Nonetheless it is desirable that all music stored in the device can be accessed quickly, and that tracks played in a sequence should match up.

In this paper, we present an approach to satisfy these constraints: a playlist containing all tracks stored in the music player is generated such that in average, consecutive pieces are maximally similar. This is achieved by applying a Traveling Salesman algorithm to the pieces, using timbral similarities as the distances. The generated playlist is linear and circular, thus the whole collection can easily be browsed with only one input wheel. When a chosen track finishes playing, the player advances to the consecutive tracks in the playlist, generally playing tracks similar to the chosen track. This behavior could be a favorable alternative to the well-known shuffle function that most current devices – such as the iPod shuffle, for example – have.

We evaluate the fitness of four different Traveling Salesman algorithms for this purpose. Evaluated aspects were runtime, the length of the resulting route, and the genre distribution entropy.

We implemented a Java applet to demonstrate the application and its usability.

1. INTRODUCTION

Imagine the following situation: A sportive person leaves the house early for her daily jogging tour. She always does sports with music, but doesn't like to listen to the same music over and over again. Thus, she has her large music collection with her on a mobile player.

She chooses the music that fits her momentary mood, without having to delay the workout: While already running, she browses through the collection with a circular wheel that could look like the one in figure 1. After playing the track she chose, the player automatically keeps on playing similar tracks. At the end of her route, she chooses on the fly some relaxing sounds while trotting home.

In this paper, we present an approach that could make this player become a reality: a music collection is organized into a large circular playlist that satisfies the constraint that on average, consecutive tracks are maximally similar. The whole playlist – and thus the whole collection – is easily accessible with only one circular controller.

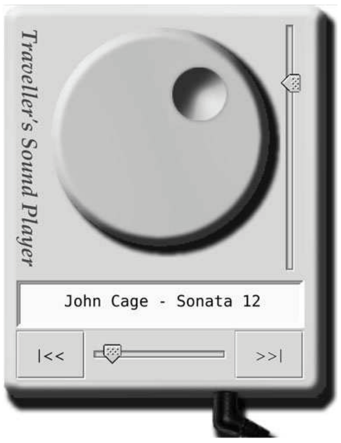


Figure 1: *Java applet demonstration: "Traveller's Sound Player".*

2. LITERATURE REVIEW

Quite some work has already been done on playlist generation:

[1] treat the problem of playlist generation as a network flow problem. Given a song collection, where each piece is labeled with a number of boolean attributes, one start track and one stop track, the algorithm finds a path (of user-defined length) through the network satisfying user-defined constraints. The proposed algorithm is an integer linear program, thus this solution is NP-hard.

In [2] a more efficient approach for handling various types of metadata is presented: according to the user-defined constraints, the metadata of each track is transformed into a cost function. The playlist construction is done by iteratively optimizing an initial randomly chosen playlist with regard to the cost function.

In [3], it is not assumed that the tracks are already labeled. The playlist generation algorithm is rather based on a music similarity function ([4]) which can be computed automatically. (In our experiments, we used a similar function.) Several approaches are evaluated for producing a playlist of given length for a given start track.

In general, for browsing music collections, there exist different approaches, for example [5], where users can find music by setting different descriptors (e.g. tempo or spectral centroid).

3. APPROACH

Musical distance data can be converted into a low-dimensional space (e.g. [6]). This low-dimensional representation can be used for playlist generation and browsing music collections. In the approach we present in this paper, we generate one circular playlist consisting of all tracks from the collection, where the goal ist to maximize the average similarity between consecutive tracks in the playlist. The resulting playlist can be interpreted as projection of the collection onto one dimension.

As most personal music collections are not labeled with detailed metadata about the musical content of the pieces, and correct metadata is not easily available, we chose to use a well-known audio based approach as a similarity measure:

For each track, on short-time audio segments Mel Frequency Cepstral Coefficients (MFCCS) are computed. The whole track is then represented as a Gaussian Mixture Model (GMM) of the distribution of MFCCS. The similarity of two tracks is the distance between their GMMs. Each of these steps is described in detail in the relevant literature (e.g. [4, 7, 8]). In our implementation, we used the parameters proposed in [8]. In previous experiments we have shown that this similarity measure outperforms many other approaches based on audio signal analysis only ([9]).

The task to generate a playlist is mapped to the Traveling Salesman Problem (TSP). The TSP is a classical problem in computer science. In its basic form, it is formulated as follows: A salesman needs to visit n cities, each of them once. The distances (or costs) to travel between the cities are known. The problem is to find the optimal (i.e. shortest or cheapest) route to visit all cities exactly once, and return to the originating city.

In our setup, the cities (i.e. nodes of a graph) correspond to the tracks in the collection, and the distances (edges) are the similarities between the tracks. Finding the optimal route means to produce a circular playlist that contains all tracks of the collection, and in which the sum of the similarities along the path is maximized.

4. ALGORITHMS

In this section, algorithmic aspects of the TSP are discussed. First, we give some general remarks, then we mention which aspects should be regarded in the specific case of similarity data, and finally we give a short description of the algorithms we evaluated.

4.1. General Remarks

The TSP is NP-hard, which implies that there is no known algorithm that calculates the exact result fast for large data sets. (For example, the state-of-the-art approach for the exact solution based on cutting planes and linear programming, needs over 90 years of CPU time[1] for 24,978 cities.)

Many heuristics have been proposed that approximate the correct result. Some of them are evaluated in this paper. We did not use an exact algorithm, as the results would hardly be different, but computation times would be much greater.

[1] 85 years to prove that the tour – found by a heuristic algorithm – is the shortest (http://www.tsp.gatech.edu/sweden/index.html)

4.2. Domain-Specific Issues

A number of heuristic TSP algorithms require the distance measure d between the nodes to satisfy the triangle inequality: $d(ac) \leq d(ab) + d(bc)$ for all triples a, b and c. The triangle inequality is not fulfilled by the similarity measure we used in our experiments, i.e. the problem is said to be *non-metric*. On our data, it did not hold in about five percent of the cases when comparing randomly chosen direct and alternative edges.

According to [10], it can be proven that for non-metric problems, it is impossible to construct an algorithm of polynomial complexity which find tours whose length is bound by a constant multiple of the optimal tour length (see also [11]). This fact applies in our scenario, as here for runtime reasons only algorithms of polynomial complexity are of interest.

4.3. Evaluated Algorithms

In the following sections, the algorithms we evaluated are described briefly. For a more detailed discussion, the interested reader is referred to the literature.

4.3.1. Greedy Algorithm

The first algorithm we evaluated is a popular simple greedy algorithm (see algorithm 1, e.g. [12]). The algorithm starts with no connected nodes. All edges are examined in increasing length. An edge is added to the initially empty set of edges if the resulting set of edges can still be combined to a valid tour.

For each number $n \geq 2$ of nodes, a TSP instance can be constructed for which this algorithm finds the *worst* possible route ([13]).

Algorithm 1 Simple Greedy Algorithm

```
1: sort edges in ascending order
2: while tour is not complete do
3:   if next longer edge can be part of a valid tour then
4:     add it to the (still uncomplete) tour
5:   end if
6: end while
```

For m edges this algorithm has the runtime $O(m \log m)$, as the most expensive step is to sort the edges in ascending order.

4.3.2. Minimum Spanning Tree

This algorithm (e.g. [14]) was evaluated although it makes the assumption that the triangle inequality is fulfilled, which is not the case on the data we used. First, a minimum spanning tree is found with a standard algorithm (Kruskal algorithm) in $O(m \log m)$, with m being the number of edges. Afterwards, a depth-first search is performed on the minimum spanning tree, and a tour is constructed by connecting the nodes in the order they are first visited during the depth-first search. (For convenience, we call this whole algorithm *MinSpan* in this paper).

4.3.3. LKH

The LKH algorithm ([11]) is an optimized version of the Lin-Kernighan algorithm proposed by Lin and Kernighan in 1971. The LKH algorithm starts with a randomly generated tour and improves it stepwise by deleting λ edges from the route and recombining the remaining tour fragments in a more efficient way. In each step, sophisticated heuristics are used to choose λ and the edges to exchange.

The runtime of the LKH algorithm is approximately $O(n^{2.2})$, with n being the number of nodes.

4.3.4. *One-dimensional Self-Organizing Map (SOM)*

To assess if an algorithm based on clustering yields better results in terms of constancy of genre membership, also a SOM algorithm was evaluated.

A SOM algorithm clusters the input data by assigning m input points to n points called *units*. During the (stochastic) clustering process, the position of each unit and the assignment of data points to units is refined iteratively. Usually, $n << m$, i.e. there are many more data points than units. SOMs have the property that they are able to project high-dimensional data into lower-dimensional spaces while preserving distance relationships to a large extent. In our experiment, we train a one-dimensional cyclic SOM, i.e. the units are arranged in a circular fashion. The one dimension corresponds to the position on the linear playlist.

Ideally, there would be as many units as tracks, so that after training, each unit would have exactly one assigned track. But as the runtime to train such a SOM is too long, we decided to use a recursive approach (algorithm 2). Before using the algorithm, the dimensionality is reduced by interpreting each column of the distance matrix as a vector and applying the principal component analysis (PCA) on them. Only the first 30 components of the PCA-compressed data were used.

Algorithm 2 Recursive Algorithm based on SOM

```
 1: train a one-dimensional cyclic SOM with $k$ units on the input points
 2: for each input point, get the best matching unit of the trained SOM
    {calculate the $k$ smaller tours ("subtours") recursively:}
 3: for each unit $u_i$ $(i = 1..k)$ do
 4:     get the input points $P_i$ belonging to the current unit $u_i$
 5:     if $|P_i| > 1$ build a tour through $P_i$ recursively
 6:     store the point (or tour) in $t_i$
 7: end for
    {combine the subtours, in a greedy manner:}
 8: break up each subtour at its longest edge
 9: get all edges that could combine two consecutive subtours, store them in $E$
10: sort $E$ in ascending order
11: while tour is not complete do
12:     if next longer $e \in E$ can be part of a valid tour then
13:         add it to the (still uncomplete) tour
14:     end if
15: end while
```

5. EVALUATION AND RESULTS

This section describes how the algorithms sketched in the previous section were evaluated: first, the data they were ran on is presented. Then, several evaluation approaches are described: For each evaluation approach, the concept of this approach is outlined, and the results obtained in our experiments are given with a short discussion.

5.1. Data Set

For computing the similarity matrix, we used 3298 tracks from the online music label magnatune[2] that were labeled with eleven genres: Ambient (4.3%), Classical (38.8%), Electronic (13.9%), Folk (1.5%), Jazz (3.2%), Metal (3.5%), New Age (5.8%), Pop (0.7%), Punk (1.9%), Rock (11.6%), World (14.8%).

5.2. Subjective Evaluation

For subjective evaluation, a Java applet was programmed that enables the user to quickly browse through the circular playlist (figure 1). Non-representative small-scale listening tests revealed that subjects liked the idea of such a sound player. In general, consecutive tracks were perceived as sounding similar. Based on the preceding tracks, users got a feel that they currently are in a certain musical region.

Also, it turned out that it is difficult to judge the quality of the generated playlists. For many playlists, users got the impression that the path is not fully straightened, i.e. it happens that musical regions are revisited after leaving them.

As each playlist consisted of over 3000 tracks, it was not possible to evaluate them exhaustively by user studies: the time to listen to a whole playlist is about one week. Thus, users listened only to short sub-paths starting at randomly chosen tracks. For evaluation based on user studies, it is also an issue that the personal impression of the playlist (which is the most important measure) is easily covered by other aspects such as if the user likes the presented music. For these reasons, the objective measurements that follow in the next sections were done.

5.3. Runtime and Route Length

For each tested algorithm, table 1 shows the absolute length of the tour, the length relative to the shortest tour found, and the runtime. For the SOM algorithm, the time to compute the PCA was not taken into account. All algorithms were implemented in MATLAB except LKH, which was given as a compiled executable[3]. The runtime of LKH was split into a preprocessing phase, which included inter alia the reading of the data from the hard drive, and the execution phase of the actual algorithm.

Algorithm	Tour Length (M)	Rel. Length (%)	Runtime (sec.)
Recursive SOM	243.7	153.0	pca + 882
Min. Span. Tree	195.3	122.6	389
Simple Greedy	172.3	108.1	520
LKH	159.3	100.0	152 + 330

Table 1: Comparison of different TSP algorithms on our similarity data. The lower bound for the tour length given by the LKH algorithm was only 0.26% better than the best tour found by LKH, and 8.14% better than the route of the simple greedy algorithm.

As expected, the most elaborate algorithm LKH had the best performance in runtime and route length. The route length was close to a lower bound also calculated by this algorithm. Surprisingly, the route calculated by the simple greedy algorithm was only about eight percent worse. The spanning tree algorithm performed

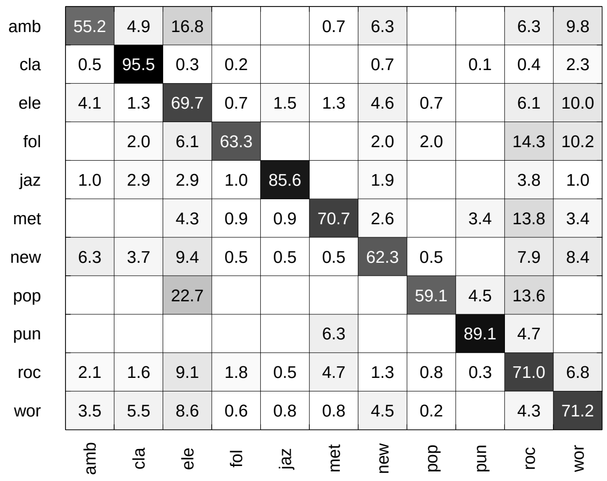
[2]http://www.magnatune.com

[3]http://www.akira.ruc.dk/~keld/research/LKH/

as expected, while the very poor performance of the SOM based algorithm is not easily explainable. Maybe, at the end of the recursion, many best matching units are only assigned few tracks, so that the presumably suboptimal greedy combination of subroutes has a degrading effect.

5.4. Fluctuations Between Genres

To assess between which genres the path changes most frequently, in figure 2 the genre memberships of pairs of tracks that follow immediately in the playlist are given for the LKH algorithm. In our experiments, we use genre memberships as an indicator, as as we assume that very similar pieces belong to the same genre.



	amb	cla	ele	fol	jaz	met	new	pop	pun	roc	wor
amb	55.2	4.9	16.8			0.7	6.3			6.3	9.8
cla	0.5	95.5	0.3	0.2			0.7		0.1	0.4	2.3
ele	4.1	1.3	69.7	0.7	1.5	1.3	4.6	0.7		6.1	10.0
fol		2.0	6.1	63.3			2.0	2.0		14.3	10.2
jaz	1.0	2.9	2.9	1.0	85.6		1.9			3.8	1.0
met			4.3	0.9	0.9	70.7	2.6		3.4	13.8	3.4
new	6.3	3.7	9.4	0.5	0.5	0.5	62.3	0.5		7.9	8.4
pop			22.7					59.1	4.5	13.6	
pun						6.3			89.1	4.7	
roc	2.1	1.6	9.1	1.8	0.5	4.7	1.3	0.8	0.3	71.0	6.8
wor	3.5	5.5	8.6	0.6	0.8	0.8	4.5	0.2		4.3	71.2

Figure 2: *Genre fluctuations for the best tour found by the LKH algorithm. Rows are the genres of the first track, and columns are the genres of the track following immediately in the playlist. Genres are: Ambient, Classical, Electronic, Folk, Jazz, Metal, New Age, Pop, Punk , Rock, World, denoted by the first three characters.*

The most frequent changes (i.e. pop following electronic, and pop following ambient) seem to be natural, as these genres are closely related.

5.5. Shannon entropy

To estimate how "locally consistent" the playlist is, the entropy of the genre distribution was calculated on short sequences of the playlists: It was counted how many of n consecutive tracks belonged to each genre. The result was normalized and interpreted as a probability distribution, on which the Shannon entropy was calculated.

The Shannon entropy is defined as

$$H(x) = -\sum_{x} p(x) \log_2 p(x) \qquad (1)$$

with $\log_2 p(x) = 0$ if $p(x) = 0$.

In figure 3, the entropy values for $n = 2..12$ are given, averaged over the whole playlist (i.e. each track of the playlist was chosen once as the starting track for a sequence of length n). 12 was chosen as the maximum length because a typical album contains about 12 tracks.

For $n = 2$, the entropy value equals the fraction of consecutive tracks that are not in the same genre. For the simple greedy and the LKH algorithm, in approximately 80% of all cases the next track is in the same genre as the current track. This number corresponds to the classification accuracy obtained when doing a 10-fold cross validation with k-NN classification for $k = 1$ (79%).

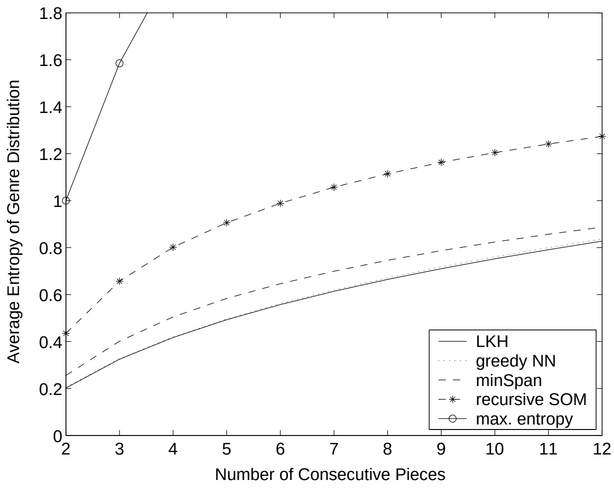


Figure 3: *Average entropy values for short sequences (length* $2..12$*) of the playlists.*

It is surprising that the entropy values of the simple greedy algorithm are only slightly worse than those of the LKH algorithm (i.e. they are almost the same). The most likely explanation is that most of the additional 8% length of the greedy route occur at genre boundaries, so that they are obscured in the genre entropy evaluation.

5.6. Long-Term Consistency

Complementary to the short-term development measured by the entropy, it is also interesting to assess the long-term development of the generated playlists. To this end, in figure 4 for each track and each genre, it is shown how many of the 75 tracks following in the playlist belong to this genre. Only the values from the playlists generated by the LKH algorithm and by the SOM based algorithm are shown.

Obviously, for the playlist generated by the SOM-based algorithm, the distribution of classical, rock and electronic pieces is less fragmented. These genres are three of the four most frequent genres. For the fourth (i.e. world) no difference is obvious.

The corresponding values for the other two algorithms which are not shown are comparable: The amount of fragmentation of the Minimum Spanning Tree algorithm was slightly worse than the amount of the SOM-based approach, and the route calculated by the greedy algorithm was even more fragmented than the route generated by LKH.

5.7. Artist Filter

In about 67%, the track that follows immediately in the playlist is by the same artist as the current track. It is a question of personal

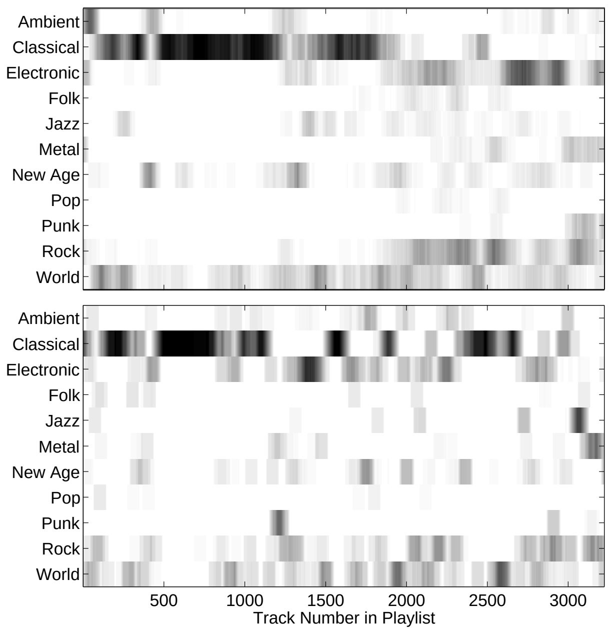


Figure 4: *Long-term distribution of the genres for the playlist generated by the SOM based algorithm (above) and the LKH algorithm (below). For each Track, it is counted how many of the following 75 tracks in the playlist have a particular genre. Full White represents 0, full black 75.*

taste if this is a desirable property[4]. If not, then a possible solution is to modify the calculated playlist. In our implementation, the playlist is checked from beginning to the end, if one of the last n tracks is by the same artist as the current track. If this is the case, the current track is delayed and inserted at the next possible position that satisfies this artist repetition constraint.

In our data set, there are on average about 13 artists per genre, but the variance is very high (for example, there was only one artist that had the genre *folk*). So, when choosing too large values for n, sections of the playlist containing only pieces from small genres could be fragmented. With these aspects in mind, we chose $n = 6$ for our experiments, i.e. tracks by the same artist have to be at least six tracks apart. During the execution of the artist filtering algorithm, there were on the average about thirty to forty tracks delayed at each step (i.e. there was a queue of thirty to forty tracks that were waiting to fulfill the artist repetition constraint).

Through the artist filtering step with $n = 6$, the tour lengths increased (see table 2). The SOM algorithm seemed most robust against artist filtering: the total tour length increased only by 52.8%, and was nearly as short as the shortest route after artist filtering. The entropy values increased also (see figure 5). Fragmentation images after filtering are not shown here, as they were only affected to a minor extent, appearing more blurred.

6. SUMMARY AND DISCUSSION

As a comparison of the different playlists by user studies was not feasible, we used several different measures to evaluate the algorithms. These tests showed:

[4]e.g. according to [3], this would be regarded as being good

Algorithm	Tour Length (M)	Rel. Length (%)	Factor (%)
Recursive SOM	372.3	100.6	152.8
Min. Span. Tree	370.0	100.0	189.5
Simple Greedy	448.9	121.3	260.5
LKH	446.9	120.8	280.5

Table 2: Artist-filtered TSP algorithms on the similarity data. Factor gives the relation of filtered to unfiltered route length (i.e. the route calculated by the SOM algorithm increased by 52.8%, while the route of the LKH algorithm increased by 180.5% when filtering artist repetitions).

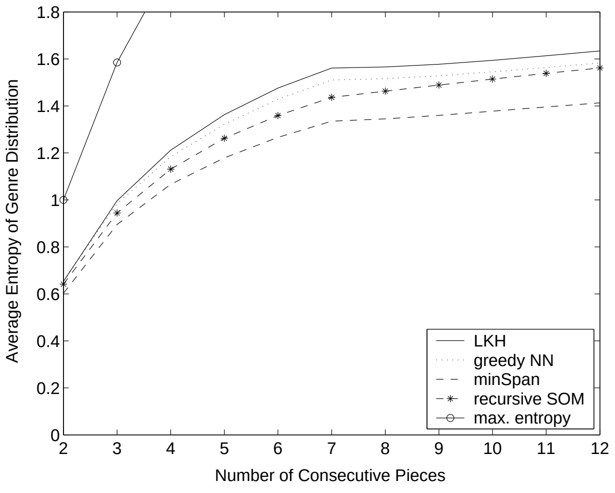


Figure 5: *Average entropy values for short sequences (length* 2..12*) of the playlists, after artist repetition filtering. Tracks of the same artist are at least 6 pieces apart, which might be a reason for the fact that entropy values in the range from 1 to 7 pieces increase faster than in the range from 7..12 pieces. Same scaling as in figure 3.*

- The LKH algorithm and the simple greedy algorithm have the advantage of fast runtime and nearly optimal route length. The small-scale genre entropy values of their routes are the best before artist filtering. But the large-scale genre distributions are quite fragmented, and their routes are vulnerable against artist filtering.
- The SOM-based algorithm had the longest runtime, the longest route length and highest entropy values before artist filtering. But it had the least fragmented long-term genre distribution, and was least vulnerable against artist filtering. After artist filtering, its route length was nearly as short as the shortest route, and the short-time genre entropy was better than those of the routes of the LKH and the greedy algorihm.
- The MinSpan algorithm had a decent runtime and was in the middle field regarding route length and entropy values before artist filtering. After artist filtering, its route length was the shortest, the short-term entropy was the best. The overall genre distribution was inferior to the one of the SOM-based approach.

The MinSpan algorithm and the SOM-based approach produce better overall genre distributions than the LKH and the simple greedy algorithm. The most likely reason for that is that locality constraints are regarded: in the SOM algorithm the overall route is built by concatenating several local routes, and in the MinSpan algorithm locality constraints are regarded as the algorithm constructs the route by a depth-first search on a minimum spanning tree.

As a final recommendation, it follows that either the SOM-based algorithm or the MinSpan algorithm seem to be favorable. The MinSpan algorithm has the advantage of faster runtime and better entropy values (both without and with artist filtering), while the route calculated by the SOM based algorithm is least affected by the process of artist filtering.

7. CONCLUSIONS AND FUTURE WORK

We presented a new approach to conveniently access the music stored in mobile sound players. The whole collection is ordered in a linear playlist and thus accessible with only one input wheel. Consecutive tracks are aimed to be consistent and maximally similar on average, thus ideally each track can be chosen as the starting point of a locally consistent playlist. Several algorithms are tested for their ability to produce such playlists.

We implemented a demonstration application for the one-dial browsing device. Such a device could offer a new way users access their music collection, as tracks are arranged according to the type of information the used similarity measure describes. The similarity measure in our implementation was timbre based, and thus the playlist is arranged by sound similarity. It may be expected that the overall results of our investigations still hold when using a different similarity measure. It is still to investigate if the noticeable drawbacks are also present with other similarity measures: sporadic intense discontinuities in the playlists, and a non-straightforward path through the collection.

The example application triggers positive reactions, and provokes the user to play around with it. In this paper, we targeted the problem of generating a playlist for such a device. As there are over 3000 pieces in the collection, with the current device it is not possible to select a desired piece precisely. It is an open question how this situation can be handled. One possibility to improve navigation would be to make only tracks that are representative for a region selectable as starting points. This is an issue of future investigations.

8. ACKNOWLEDGMENTS

This research is funded by the EU FP6 project 507142 SIMAC ("Semantic Interaction with Music Audio Contents"). The Austrian Research Institute for Artificial Intelligence also acknowledges the financial support of the Austrian Federal Ministries of Education, Science and Culture and of Transport, Innovation and Technology.

9. REFERENCES

[1] M. Alghoniemy and A. Tewfik, "A network flow model for playlist generation," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'01)*, Tokyo, Japan, August 22-25 2001.

[2] J.-J. Aucouturier and F. Pachet, "Scaling up music playlist generation," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'02)*, Lausanne, Switzerland, August 26-29 2002.

[3] B. Logan, "Content-based playlist generation: Exploratory experiments," in *Proceedings of the International Conference on Music Information Retrieval (ISMIR 2002)*, Paris, France, October 13-17 2002.

[4] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'01)*, Tokyo, Japan, August 22-25 2001.

[5] G. Tzanetakis, "Musescape: An interactive content-aware music browser," in *Proceedings of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, London, UK, September 8-11 2003.

[6] E. Pampalk, A. Rauber, and D. Merkl, "Content-based organization and visualization of music archives," in *Proceedings of the ACM Multimedia*, Juan les Pins, France, December 1-6 2002, pp. 570–579, ACM.

[7] J.-J. Aucouturier and F. Pachet, "Music similarity measures: What's the use?," in *Proceedings of the Third International Conference on Music Information Retrieval (ISMIR'02)*, Paris, France, October 13-17 2002, pp. 157–163, IRCAM.

[8] J.-J. Aucouturier and F. Pachet, "Improving timbre similarity: How high is the sky?," *Journal of Negative Results in Speech and Audio Sciences*, vol. 1, no. 1, 2004.

[9] T. Pohle, "Extraction of audio descriptors and their evaluation in music classification tasks," M.S. thesis, Technische Universität Kaiserslautern, Austrian Research Institute for Artificial Intelligence (ÖFAI), Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 2005.

[10] S. Sahni & T. Gonzales, "P-complete approximation algorithms," *Journal of the Association for Computing Machinery*, vol. 23, pp. 555–565, July 1976.

[11] K. Helsgaun, "An effective implementation of the linkernighan traveling salesman heuristic," DATALOGISKE SKRIFTER (Writings on Computer Science) No. 81, Roskilde University, 1998.

[12] D.S. Johnson and L.A. McGeoch, *Local Search in Combinatorial Optimization*, chapter The traveling salesman problem: A case study in local optimization, pp. 215–310, Wiley, New York, 1997.

[13] G. Gutin, A. Yeo, and A. Zverovich, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp," *Discrete Applied Mathematics*, vol. 117, pp. 81–86, 2002.

[14] S. S. Skiena, *The Algorithm Design Manual*, Springer-Verlag, New York, Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, 1997.