An Approach to Automatically Tracking Music Preference on Mobile Players

Tim Pohle,[1] Klaus Seyerlehner[1]
and Gerhard Widmer[1,2]

[1] Department of Computational Perception
Johannes Kepler University Linz, Austria
[2] Austrian Research Institute
for Artificial Intelligence (OFAI)
Vienna, Austria
music@jku.at

Abstract. More and more music is being made available to the music listener today, while people have their favorite music on their mobile players. In this paper, we investigate an approach to automatically updating the music on the mobile player based on personal listening behavior. The aim is to automatically discard those pieces of music from the player the listener is fed up with, while new music is automatically selected from a large amount of available music. The source of new music could be a flat rate music delivery service, where the user pays a monthly fee to have access to a large amount of music. We assume a scenario where only a "skip" button is available to the user, which she presses when the currently playing track does not please her. We evaluate several algorithms and show that the best ones clearly outperform those with lower performance, while it remains open how much they can be improved further.

1 Introduction

Today, portable music players and mobile communication devices are more and more merging. For example, many mobile phones have audio players built into them. Thus, there is the possibility that people obtain new music by using a service over the air. The idea to offer such a service in the form of a music flat rate has been around for a while (be it over the Internet or immediately over the phone's data connection). Some flat rate music delivery services are being planned, and others are already available. For example, services are being offered by Yahoo[3], Microsoft[4], Napster[5], and diverse mobile communication providers, such as One[6]. The user pays a fixed monthly fee and can listen to music with little

[3] http://music.yahoo.com
[4] http://music.msn.com
[5] http://www.napster.de
[6] http://www.one.at

or no limitations with respect to the number of tracks. Some services include the option to transfer the tracks to mobile players for listening to them offline.

In such a scenario, it might be beneficial to support the user with MIR based techniques. Two aspects come to mind: First, the user may be supported in selecting those tracks she likes from all those that are available over the service. Second, as the amount of storage is eventually limited on any device, the user may be supported in selecting those tracks she wants to be deleted from the player (or moved to a larger persistent storage). The ultimate idea of such a MIR application is to keep the collection on the user's device up-to-date at any point in time, with only little effort (and/or interaction) by the user.

In this work, we present algorithms aimed to accomplish such a task based on an audio similarity measure [AP04,ME05]. Using user feedback to select matching tracks has been suggested in [PPW05,Pam06], from which we adopt ideas for track selection algorithms.

The remainder of this paper is organized as follows: After a brief literature review in Section 2, Section 3 gives a detailed and formalized description of the suggested application. The proposed algorithms are described in Section 4 and results presented in Section 5. Conclusions are drawn in Section 6.

2 Related Work

A number of MIR systems have been developed to support users in finding music[7]. Some of them e.g., [VP05,PALB+06], use several data sources and make use of metadata associated with the music. While this helps in supporting robustness, when considering a user's music collection, metadata may be not available, or its quality may not always be assured. For example, different sources of metadata may assign similar music into different categories. Based on such considerations, we assume that metadata is not available, but rather one music similarity measure based on the audio signal captures the similarity of the songs. The approach is based on an audio similarity measure that uses the well-known Mel Frequency Cepstral Coefficients (MFCCs) as a basis for similarity computation [AP04,ME05]. While such algorithms widely are evaluated by genre classification experiments, the one advantage of using them is that they do not assign a track to one (or more) classes, but rather one yields a usually continuous similarity score between each pair of songs. For example, by doing so, similar-sounding music that one usually would assign to different genres still can be considered as being similar. In [PPW05], a scenario is presented where similarities between all tracks on a user's player are known (or calculated from the audio data), and a playlist should be generated on the fly based on the user's current (dis)likes. To support mobile use, the user interface is very limited. Tracks to be played are selected based only based on the user's presses of a skip button. Several algorithms are suggested and compared, and the best performing seem to perform good enough for use in a real-world scenario. We adopt the ideas of these algorithms for the scenario presented here. Otherwise, the scenario described here is

[7] an overview can be found at http://mirsystems.info/

more complicated than the one in [PPW05], where each track is either played or skipped. Here, a particular track might be played and / or skipped multiple times. Furthermore, there is not only the decision which track should be played (or added to the player), but also which tracks the user does not like any more, and consequently should be removed from the player.

For evaluation of MIR techniques, generally the best approach is to conduct user evaluations (as e.g. in [VP05]) or surveys. As user studies are cost and time intensive, and our work is still in an experimental state, we follow the approach from [PPW05] and restrain to automated experiments where a certain user behavior is assumed.

3 Application Scenario

The basic usage scenario in the suggested application is as follows: When listening to music that is on her mobile player while being en route, the user skips those tracks she currently does not like. When returning home, she plugs the player to the personal computer. At this point, the application selects those tracks that should be removed from the player because the user does not like them so much any more. These tracks are transferred to the computer's hard disk. Also, the application selects those tracks available over the music flat rate that the user is likely to like. Those tracks are obtained from the service and transferred to the device. More formally, the scenario is as outlined in the following paragraphs.

3.1 Initial State

In the scenario of this paper, the user likes all tracks that are initially on the device. It is assumed that the user only adds music she likes to the device when she has just obtained it, which seems a quite reasonable assumption.

3.2 Listening Session

After the user has copied music she likes to the device, she uses the device to listen to music. It is assumed that during such a listening session the music on the device remains the same, i.e., there is no active connection to the music repository. This assumption is made for several reasons.

- The device itself may not have a connection (e.g., a rather simple stand-alone music player). If the device does have connectivity, then circumstances may prohibit its use (e.g., on airplanes connectivity may not be used). For these reasons, assuming that connectivity is not available makes the modeling more general.
- Assuming a device that is always connected to the music database (repository), the scenario is much closer to the one presented in [PPW05], because basically all tracks are available for streaming. Thus, the problem reduces to selecting a track that currently best fits. However, even in this scenario, it might be beneficial to know which tracks the user has disliked in the past and thus is likely to dislike also currently.

An important aspect of a listening session is how many tracks k_i the user actually listens to in listening session i before the session ends and the device is being updated. In the experiments presented here, this is a random number in the range from 15 to 25, corresponding to approximately 45 to 75 minutes of music. For better comparability between parameter settings, k_i is the same for each experiment (however, not the same for each i).

Another important factor is how the next track is selected that is played to the user. There are at least three possible algorithms:

1. The tracks are presented in a fixed predetermined order, e.g. in alphabetical order or in the order they were added to the device.
2. An algorithm based on skipping behavior is used during the listening session [PPW05].
3. Tracks are presented in random order.

If the track update algorithm works well, then most tracks on the device fit the user's taste and thus are acceptable for the user. However, when using fixed presentation order (case 1), the first tracks may become boring to the user, provoking the user to skip the first tracks, which adds false skips (i.e., skips that do not indicate a genuine dislike). Presenting tracks in random order (case 3) is an algorithm implemented in most current audio players, i.e., a de facto standard. This, and the fact that using an algorithm based on skipping behavior (case 2) might induce unwanted effects on the number of skips (e.g., such an algorithm may prefer or avoid certain tracks), makes it the algorithm of choice for the experiment.

3.3 User Feedback

There are several ways one could think of for the user to give feedback about her liking or disliking of the currently played music. The most simple user interface would consist only of a *skip* button the user presses if she dislikes the currently played track. Not pressing this button is interpreted as the user liking the played music.

However, in a real-world application, there might be a variety of reasons why the user presses the skip button. Besides accidental presses, most notably the user might press the button because she does not want to listen to this track right now (but in general she likes the track). Thus an algorithm based on such feedback data needs to be robust against such influences. To avoid ambiguities, it is possible to add a second button with which the user can indicate that she wants to *ban* the currently played track. In this scenario, pressing the *skip* button then has no particular meaning to the track discard algorithm. In the experiments to be described below, we simulated both skipping and banning.

3.4 User Modeling

For a systematic quantitative experimentation, we need to simulate a hypothetical user. Specifically, we need to define a criterion by which the user decides

whether or not she likes a piece or hits the skip/ban buttons. We assume two main use cases, comparable to [PPW05]:

- *Use Case 1:* The user only likes tracks from one genre. Generally, tracks from this genre are accepted, and tracks from all other genres are skipped.
- *Use Case 2:* In the beginning, the user only likes tracks from one genre (A). Over time, the user's preferences change to a different genre (B).

In the second case, tracks from genre A are accepted in the first $\frac{3}{4}$ of the listening sessions, and tracks from genre B are accepted during the last $\frac{3}{4}$ of the listening sessions.

Also we take into consideration that once the user has listened to a particular track many times, the track gets boring, and eventually the track is (almost) always skipped (cf. [CDB05]). In the use cases, this is formalized as a number $\texttt{maxListen}_t$ that is associated with each track t. Once the (hypothetical) user has accepted track t this often, she will never accept it again (i.e., always skip it afterwards). The value of $\texttt{maxListen}_t$ is chosen randomly in the range from 10 to 20 for each track and kept fixed for all different experiment settings.

Obviously, the algorithm is also influenced by the number of tracks that fit onto the device (denoted as `capacity`). Today, most mobile players have at least 1 GB of memory, which corresponds to about 140 tracks in high encoding quality (if a track has five minutes length and a bit rate of 192 kbit per second). A realistic alternative figure would be 4 GB (555 tracks). For the experiments, it is of importance to consider the relation of the number of tracks that fit on the device and the number of tracks that are available in the "repository" (i.e., the tracks available over the flat rate service). Particularly the latter is limited by the size of the music collection available for the experiments.

Finally, it is important to model unreliability of the user's input in the experiments: if the input is fully reliable, then pressing the skip button for a piece means that the skip button will always be pressed for this track (in the current experiment). Consequently, an algorithm can safely discard all tracks for which the skip button has been pressed once. As this is unrealistic, unreliability of the input is simulated by reversing the user's decision to press the skip button (or not to press it) with a certain probability. In our experiments, we use $p = 0.2$. The *ban* button, on the other hand, is always deterministically pressed when the simulated user does not like a piece (or does not like it any more), according to the present use case.

All these discussed parameters have to be considered when evaluating the algorithms. Lacking a basis, we set most of them in an ad-hoc manner, assuming that the relative performance of the evaluated algorithms with respect to each other will persist over a range of possible parameter settings. The evaluated algorithms are presented in the next section.

4 Evaluated Algorithms

As already indicated, the algorithm mainly consists of two parts: When the device is updated, first those tracks are selected that are not of interest to the user

any more. These tracks are removed from the device, and replaced by tracks selected in the second stage of the algorithm. Obviously, the first part of the algorithm (the *discard policy*) has a limiting effect on the potential benefit of the second part (the *track selection policy*). If the discard policy performs poorly, e.g., by removing too few outdated tracks from the device, then the track selection policy can not replace these tracks with better matching ones.

4.1 Discard Policy

For creating discard policies, it is of use to define a basic measure of acceptance called *like value* here. Based on the skip count s_t and the listen count l_t of track t, the like value of track t is

$$\mathtt{lv}_t = \frac{l_t}{l_t + s_t} \tag{1}$$

The like value is only defined for tracks that have already been presented to the user.

The following discard policies were evaluated. The first of them only assume the presence of a skip button, while the last is based on the ban button.

1. *skipped_once.* Discard *all* tracks that have been skipped. This is the algorithm that discards most tracks. Particularly, also tracks that have only accidentally been skipped are immediately discarded. This is formalized in the like value above: discard tracks with a like value lower than 1.0.
2. *skip_larger_accept.* Discard tracks that are more often skipped than accepted. Formally: discard tracks with a like value lower than 0.5. Obviously, this discard policy is much more conservative than policy 1.
3. *like_value.* As discard policy 2 might be too conservative, and 1 might discard too many tracks, in the third evaluated discard policy, a track is discarded if it has a like value lower than 0.75.
4. *userbanned.* The most informative discard policy is to discard exactly those tracks that are known to be not liked by the user (any more). This is the best possible policy, but comes at the cost of having an additional button on the device.

In a realistic setting, the user should be able to mark a track "sticky", so that it is not removed from the device. Here, this is omitted as such an option would further complicate the experimental setup.

4.2 Track Selection Policy

To decide which songs to add to the player, the system relies on an *audio similarity measure* that compares tracks on the mobile device to tracks in the repository. More specifically, we use the following similarity measure here: Each track is represented as a Gaussian model of Mel Frequency Cepstral Coefficients (MFCCs) calculated on short audio frames. Two songs are compared by calculating the KullbackLeibler (KL) distance between their models. More detailed descriptions of the approach can be found in the literature ([AP04,Pam06]).

Based on this, the following track selection policies were evaluated (cf. [PPW05]):

1. *Random Baseline.* Added tracks are selected from the "candidate" tracks in the repository in a random manner.
2. *Policy A.* The order in which new tracks are added to the device is determined only based on the tracks that are initially on the device. All tracks in the repository are ordered once by their minimum distance to any of the tracks initially on the device.
3. *Policy B.* Those songs closest to any of the songs listened to during the last listening session are added.
4. *Policy C.* Select songs closest to any of the last `capacity` tracks[8] that have been accepted (i.e., listened to).
5. *Policy D.* Select songs based on the last `capacity` tracks that have been accepted (set A) and the last `capacity` tracks that have been skipped (set R). Let d_a be the distance of song d to the closest song from A, and d_r the distance from song d to the closest song in R. From all tracks with $d_a < d_r$, select those with smallest d_a and transfer them to the player. If these are not enough tracks to fill the player, go on with adding those tracks with smallest $\frac{d_a}{d_r}$. Optionally, the distance calculation is weighted by the number each track in A and R was listened to or skipped, respectively.

To determine the distance of a candidate track in the repository to the tracks on the device (i.e., the subset used to determine the distance), the minimum distance of the track to all tracks of interest is used [Log04]. Policies B, C and D were also evaluated in a second variant, where distances to the tracks on the device were weighted according to the number of times the corresponding tracks have been accepted. Distances to tracks that have more frequently been listened to were weighted higher. In this case, the final distance is the weighted mean distance instead of the minimum.

For all policies the general rule applies that a song that already has been discarded from the device in the past will not be added again. Of course, in a real-world application, the user should be allowed to do so manually, which could eventually improve the quality of the suggested songs.

5 Evaluation Results

To measure the quality of an algorithm, two measures were used: the average number of times the skip button was pressed by the (simulated) user, and the number of tracks discarded by an algorithm. In both cases, generally lower means better. However, of course if too few tracks are discarded, then the number of times the skip button is pressed increases.

For *userbanned*, it is of interest how often skip or ban was pressed, both are counted together. The skip button presses are also counted to allow a comparison to the other policies, i.e., to also take into account the effects of the random factor p that is only applied to the skip button, but not to the ban button. This results in the reasonable situation where each button press is counted as a user

[8] or fewer, if fewer than `capacity` tracks have been accepted so far

interaction. For the other track discard policies besides *userbanned*, it is only of interest how often skip was pressed, as ban presses are ignored by the other policies. Some parameters used in the experiments are given in Table 1.

Parameter	Value
`capacity` (# of tracks on device)	100
p (randomly invert skip decision)	0.2
# listening sessions	100
k_i (# of tracks listened to in session i)	15 to 25 (randomly chosen for each i)
$\texttt{maxListen}_t$	10 to 20 (randomly chosen for track t)

Table 1. Parameters used in the experiments.

5.1 Data

We evaluated the various approaches on a music collection consisting of $51,056$ tracks by $7,610$ artists. Tracks were assigned to 45 genres. For the experiments, the 18 largest genres were used (while all tracks were left in the collection). Small genres were disregarded because potentially they did not contain enough tracks to offer enough matches (e.g., in cases where the discard policy would discard very many tracks), which could have resulted in artificially low evaluation results. All used genres consisted of more than $1,500$ tracks each. Altogether, the 18 largest genres contained $35,265$ tracks. For evaluation, each of the 18 genres was assumed to be "accepted" music in turn, and results were averaged.

5.2 Use Case 1

Figure 1 gives the average number of button presses that sum up during the considered simulated usage time of the device (100 listening sessions) when only tracks by a particular genre are accepted. For better comparability, values are given relative to the best obtained value of the discard policy *userbanned*, which is thought to give an indication of an upper bound. On average over all policies, $1,933$ out of a total of $3,909$ suggested tracks were fully listened to by the hypothetic user during all listening sessions.

The results show that even for the better-performing (non-baseline) algorithms, overall in about 50% of the cases a suggested song was skipped. This seems a large number, but it should be considered that even for a perfect algorithm this would still be 20% because of the random factor. To evaluate if these results are low enough for a real-world application, a user study is necessary.

A closer look at the relative performance of the algorithms shows that as expected, the best discard policy is *userbanned*. The other discard policies produce much higher values, starting at a factor of 1.8 times as high.

For the track selection policies, it is somewhat surprising that the simple *random* policy which was used as a baseline does not always perform worse

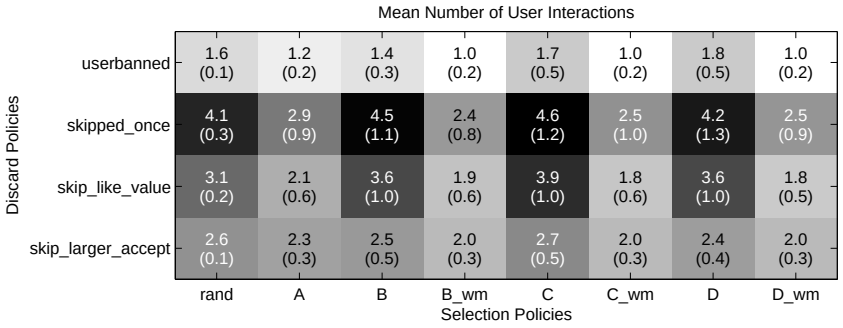

Fig. 1. UC1: Mean and standard deviation of number of *user interactions* (button presses), summed over 100 listening sessions. Mean and standard deviation measure this value over 18 genre experiments. Values are normalized with respect to the best (minimum) average value obtained for *userbanned*, which is 1058.5.

than more sophisticated approaches (B, C, D). However, all of B, C and D are greatly improved when the similarity measure is weighted according to the number of times a song has actually been listened to (B_wm, C_wm and D_wm). In combination with *like_value*, they produce the best results achieved in the experiments in cases where only a skip button is available. Taking into account the number of times a song was accepted seems an important step, resulting in the best values. A likely reason is that the random factor is leveled out to a certain degree. Also it is interesting to note that all three algorithms that use weighted values perform quite similarly, which may be an indication that this is some sort of upper bound for the examined techniques.

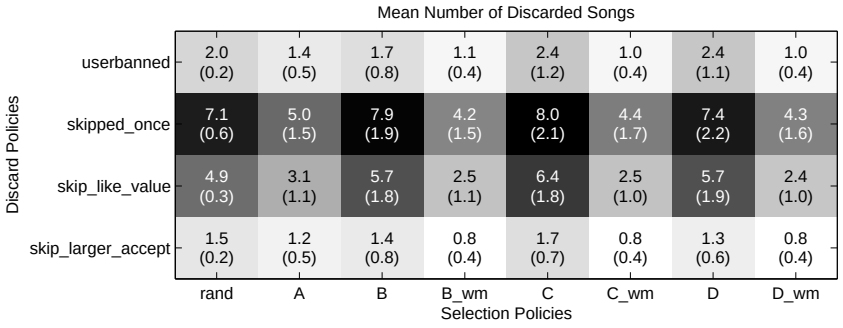

Fig. 2. UC1: Mean and standard deviation of number of *discarded songs*, measured relative to the minimum mean number of user interactions for *userbanned*, which is 607.1.

To complement the number of button skips, Figure 2 gives the number of songs that were removed from the device (and replaced with new tracks) in the

same period. As expected, using the discard policy that discards tracks most reluctantly (*skip_larger_accept*) leads to the lowest number of discarded tracks. These numbers are even lower than for *userbanned*. Also, it is of interest that the number of discarded songs decays from *skipped_once* over *skip_like_value* to *skip_larger_accept*, while the number of user interaction was lowest for *skip_like_value*, which we see as an indication that this is the most recommendable of the presented *skip* policies.

5.3 Use Case 2

Figures 3 and 4 give the corresponding data based on use case 2, when the user's preferences change over time (i.e., the genre accepted by the user shifts from a genre A to genre B). Each of the 18 genres was used as the starting genre A in turn. A corresponding genre B was chosen manually, where each genre appeared once as genre B. Most transitions were between somewhat related genres, although there also was the unusual transition *Classical* to *Pop-Rock*. Generally, the results show the same tendency as for use case 1. Again, the combination of track discard policy *skip_like_value* with selection policy *D_wm* resulted in the lowest values in the non-baseline scenario where only a skip button is available, although there is no big difference to the other two algorithms based on weighted values.

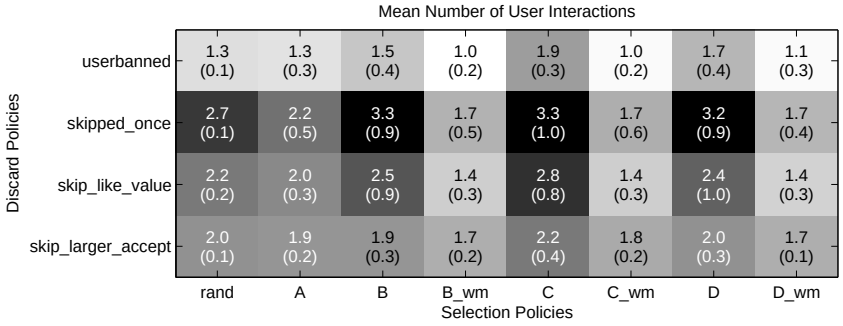

Fig. 3. UC2: Mean and standard deviations of number of *user interactions* (button presses), measured relative to the minimum mean number of user interactions for *userbanned*, which is 1383.

6 Conclusion

We have presented several algorithms aiming to automatically select music that is on a user's portable music player. The algorithms assess the user's like and dislike of individual tracks by the number they were skipped. Tracks the user

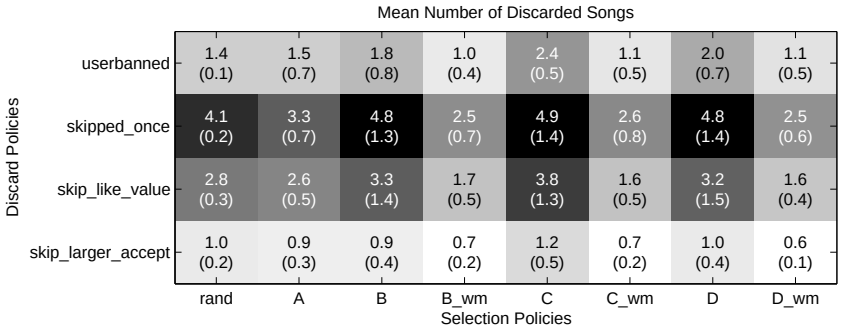

Fig. 4. UC2: Mean and standard deviations of number of *discarded songs*, measured relative to the minimum mean number of discarded songs for *userbanned*, which is 928.5

seems to dislike are removed from the device, and replaced by tracks that seem to fit the user's current music taste. Our automated experiments showed clear differences between the suggested algorithms, while weighting tracks based on the number they were skipped improves the examined algorithms to a comparable level with regard to the evaluation measures. User studies could show if the algorithms are sufficiently good for using them in a real world application, and to measure the actual user experience.

References

[AP04] J.-J. Aucouturier and F. Pachet. Improving timbre similarity: How high is the sky? *Journal of Negative Results in Speech and Audio Sciences*, 1(1), 2004.

[CDB05] Sally Jo Cunningham, J. Stephen Downie, and David Bainbridge. "The Pain, the Pain" : Modelling Music Information Behavior and the Songs we Hate. In *Proceedings of the* 6^{th} *International Conference on Music Information Retrieval (ISMIR 2005)*, London (UK), September 11-15 2005.

[Log04] Beth Logan. Music Recommendation From Song Sets. In *Proceedings of the* 5^{th} *International Conference on Music Information Retrieval (ISMIR 2004)*, Barcelona (Spain), October 10-15 2004.

[ME05] Michael Mandel and Dan Ellis. Song-Level Features and Support Vector Machines for Music Classification. In *Proceedings of the* 6^{th} *International Conference on Music Information Retrieval (ISMIR 2005)*, London (UK), September 11-15 2005.

[PALB+06] F. Pachet, J.-J. Aucouturier, A. La Burthe, A. Zils, and A. Beurive. The cuidado music browser : An end-to-end electronic music distribution system. *Multimedia Tools and Applications*, 33(1):331:349, 2006. Special Issue on the CBMI03 Conference.

[Pam06] E. Pampalk. *Computational Models of Music Similarity and their Application in Music Information Retrieval*. Docteral dissertation, Vienna University of Technology, Austria, March 2006.

[PPW05] E. Pampalk, T. Pohle, and G. Widmer. Dynamic Playlist Generation Based on Skipping Behaviour. In *Proceedings of the* 6^{th} *International Conference on Music Information Retrieval (ISMIR 2005)*, London (UK), September 11-15 2005.

[VP05] Fabio Vignoli and Steffen Pauws. A Music Retrieval System Based on User-Driven Similarity and its Evaluation. In *Proceedings of the* 6^{th} *International Conference on Music Information Retrieval (ISMIR 2005)*, 2005.