Case-based Sequential Ordering of Songs for Playlist Recommendation*

Claudio Baccigalupo and Enric Plaza

IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (Spain)
Vox: +34-93-5809570, Fax: +34-93-5809661
Email: {claudio,enric}@iiia.csic.es

Abstract. We present a CBR approach to musical playlist recommendation. A good playlist is not merely a bunch of songs, but a selected collection of songs, arranged in a meaningful sequence, e.g. a good DJ creates good playlists. Our CBR approach focuses on recommending new and meaningful playlists, i.e. selecting a collection of songs that are arranged in a meaningful sequence. In the proposed approach, the Case Base is formed by a large collection of playlists, previously compiled by human listeners. The CBR system first retrieves from the Case Base the most relevant playlists, then combines them to generate a new playlist, both relevant to the input song and meaningfully ordered. Some experiments with different trade-offs between the diversity and the popularity of songs in playlists are analysed and discussed.

1 Introduction

A typical music recommender considers songs the user likes, has listened to, or has bought, and proposes similar songs that the user will probably like, be interested in listening, or buying. Only a few recommenders are addressed to suggest *playlists of similar songs* with an inherent *sequential structure*.

Creating a playlist is a common manual operation in audio user experience. Imagine a user wants to assemble a playlist, on the basis of a specific song he likes, has listened to or has bought. He could either choose each song manually, or follow the advice of a recommender, adding one proposed song after the other. In both cases, the process would be slow; besides, the user would also have to *order* the songs, for a playlist is expected to contain songs in a specific sequence.

In this paper, we describe a case-based approach to playlist recommendation. Given a song chosen by the user, we don't focus on finding isolated *similar songs*; instead we put the emphasis on recommending *good playlists*. A playlist is a sequence—not a set—of songs; thus its quality is given both by the songs it contains, and by their ordering (their relative positions).

* This research is supported in part by a MusicStrands scholarship and by CBR-ProMusic under the project TIC2003-07776-C02-02.

This paper presents a CBR system that takes a song as the input and returns a recommended playlist as the output. The possible applications include: personalised radio programs, playlist generators, digital music organisers, and in general any scenario where the user desires a good sequence of songs, related to a chosen one, without the burden of compiling it manually.

In this paper, after reviewing related works (Sect. 2), we present the problem of recommending a playlist from a song (Sect. 3). Then we describe the CBR process: in Sect. 4, we illustrate the structure of the Case Base; in Sect. 5, we explain the Retrieve process, and introduce the basic concepts of pattern and relevance; in Sect. 6, we describe how the retrieved playlists are combined in the Reuse process. In Sect. 7 we report about the tests with an actual Case Base of playlists. Section 8 ends the paper with a brief conclusion and future works.

2 Related Work

Several approaches have been investigated to develop an Automatic Playlist Generator [2, 7, 17, 3, 20, 16]. Some of them require a large pre-existent *music-related* knowledge, either in form of music metadata [18] or acoustic-based measures [12]. In this paper we introduce a 'knowledge-light' approach to recommendation, based only on *user-related* knowledge. In this sense, our approach is comparable with SmartRadio and CoCoA Radio, two systems with which we share the vision that the work involved in compiling a playlist of music can be distributed to other listeners [10]. SmartRadio [8] is an Automatic Collaborative Filtering recommender that generates personalised playlists on the basis of playlists of users whose profiles are similar. Similarly, CoCoA Radio [4] recommends new playlists, but it also assumes that the knowledge of a playlist is encoded in the order of its songs. Our system is, in a sense, *lighter* than SmartRadio and CoCoA Radio. We also do assume that, in any well chosen collection of music, there is some great *implicit* value in the order of its songs, but we use this and only this knowledge to generate new playlists, without, for instance, collecting users' profiles or managing *explicit* ratings.

Some common points can be found between our system and the approach presented by Ragno et Al. [21]. In their work, similarity is inferred analysing the occurrences of songs in broadcasted streams of music, and new playlists are generated using a *graph-search process* (where the songs are the nodes of the graph and the similarities between them represent the weights of the arcs).

In our approach, the recommender is implemented as a *Case-Based Reasoning* (CBR) process [1, 14]. Every playlist is seen as a case whose relevance is inferred measuring the *co-occurrences* of its songs in a large collection of past playlists. Co-occurrences analysis [11] has been proved effective in many domains where information is available as large sets of sequential data. Applications range from semantical similarity recognition [13] to weather forecast [22]. Pachet et al. [15] have originally proposed co-occurrences analysis as a method to evaluate similarity between songs. In this paper, we combine this method with an explicit notion of sequentiality, in order to infer similarity between songs and playlists.

3 Playlist Recommendation

In our approach, the user chooses an input song s and a desired length λ; the goal of the recommender is to return a playlist p (with λ songs) such that: **(G1)** p contains s; **(G2)** p is varied; **(G3)** p is coherently ordered. A playlist is *varied* if it does not repeat the same track or artist in the sequence, or at least if repetitions are not close. We say a playlist is *coherently ordered* if its songs make sense in the proposed sequence. The problem we have to solve is then to determine *which sequences of songs are coherently ordered*.

In order to answer this issue, we analyse a large repository of playlists, previously compiled by human listeners. In this analysis, we seek for those sequences (of two or more songs) that occur more often, with the same order, in these playlists. The idea is that the higher the number of playlists where a sequence occurs, the stronger the evidence that sequence is coherently ordered. This idea takes inspiration from studies of *co-occurrences analysis*. Basically, these studies argue that when two items appear frequently in the same context, this proves the existence of some similarity between them. For instance, linguists regularly use co-occurrences analysis to extract clusters of semantically related words from large collections of texts. We adapt this approach to a musical context: if some songs occur frequently, in the same order, in a collection of past playlists, then we assume that the order in which they appear is meaningful. We call such sequences of songs *relevant patterns*.

Relevant patterns are the starting point for the recommender we present; to fulfil the aforementioned goals we employ a Case-Based Reasoning approach that comprises four subsequent steps:

1. Assemble a Case Base from a repository of playlists *(Case Base Setup)*.
2. Prompt the user for an input song s and the desired length λ of the recommendation *(Problem Description)*.
3. Retrieve from the Case Base a subset of playlists that satisfy most of the goals. Specifically, the retrieved playlists are varied **(G2)** and include relevant patterns that contain s **(G1)** *(Retrieve Process)*.
4. Combine the songs of the retrieved playlists to generate a coherently ordered playlist **(G3)** with length λ *(Reuse Process)*.

Notice that the knowledge that guides the CBR process comes from the same content of the Case Base, in the form of relevant patterns. Hence the quality of the recommendation depends mostly on the quality of past playlists: the more accurately they have been compiled by the users with a meaningful order, the more this order will be reflected in the output. Clearly, different repositories may lead to different recommendations.

4 The Case Base

Let $\mathcal{P}$ be the repository of past playlists. $\mathcal{P}$ may include, for example, tracklists from radio programs, web streams, music compilations, DJ sessions, and

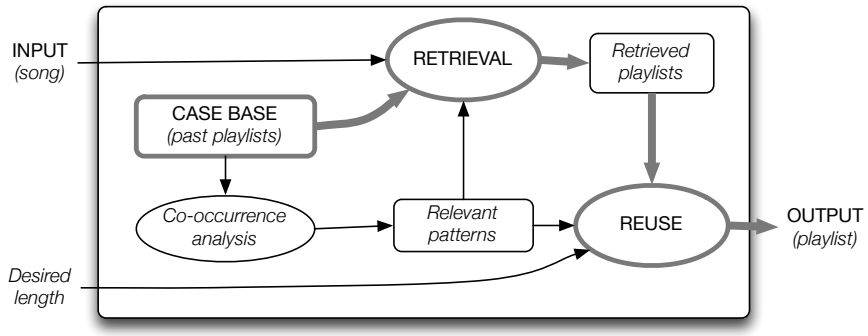

Fig. 1. General view of the CBR process.

in general lists of songs that have been compiled with a meaningful order. By the way, the repository could include some *noisy* playlists, that is, playlists that have not been compiled with a great accuracy, and that would lower the quality of the Case Base if used. Examples of noisy playlists are: very long sequences of songs (that are probably only large *sets* of songs, with no meaningful order), very short sequences (from which we cannot extract any relevant pattern), and sequences in alphabetical order (that is hardly a meaningful order). Before using the repository, we filter $\mathcal{P}$ to remove such playlists; the result $\mathcal{C}$ is the Case Base.

Every playlist of $\mathcal{C}$ is a sequence of songs: $p = (s_1, s_2, .., s_n)$. Let $\Lambda(p) = n$ be the length of p. A song s is contained in a playlist p if the following condition holds: $In(s, p) = \exists i, 1 \leqslant i \leqslant n : s = s_i$

Every song is characterised by a set $\mathcal{A}$ of attributes (e.g.: an identifier of the track, an identifier of the album it belongs to, the artist that performs it, etc.). We assume that the user will disapprove of recommendations where close songs repeat the same attributes: for example playlists where many songs belong to the same album or artist. Such playlists lack necessary variety, and our approach intends to avoid these cases, in order to boost diversity. For every attribute $a \in \mathcal{A}$, we introduce a parameter γ_a, called *safe distance for* a, as the minimum number of songs that must occur before we allow a value of a to be repeated. For instance, if $\gamma_a = 10$, the same value of a can be repeated without restriction only after 10 songs. To quantify how many, and how close, are the potential repetitions of an attribute in a playlist, we introduce a function called *attribute variety*:

Definition 1. ***(Attribute variety)*** *Let $p \in \mathcal{C}$ be a playlist, $p = (s_1, s_2, .., s_n)$, and $a \in \mathcal{A}$ be a song attribute; the* variety of a in p *is a function $V_a : \mathcal{C} \rightarrow (0, 1]$*

such that:

$$V_a(p) = \prod_{i=1}^{n} \begin{cases} \frac{j-i}{\gamma_a} & \textit{if } \exists j, i < j \leqslant \min(n, i+\gamma_a) : a(s_i) = a(s_j) \\ & \wedge \forall k, i < k < j, a(s_i) \neq a(s_k) \\ 1 & \textit{otherwise} \end{cases}$$

In other words, a playlist has an attribute variety $V_a(p) = 1$ only if no value of a is repeated within the safe distance γ_a; otherwise the more and the closer the repeated values, the smaller $V_a(p)$. Combining the values of $V_a(p)$ for all the song attributes with a t-norm (e.g. product), we obtain a global measure of *variety of a playlist* $p \in \mathcal{C}$:

$$Var(p) = \prod_{a \in \mathcal{A}} V_a(p)$$

The variety of a playlist p equals 1 only if no attribute is repeated in p within its safe distance. According to **(G2)**, we are interested in recommending this kind of playlists. We will see in Sect. 5 how $Var(p)$ will be used, inside the Retrieve process, to prefer varied playlists over repetitive ones.

5 Retrieve

In a classic CBR approach, the goal of the retrieval task is to return a subset of cases that contain the *solutions* for *problems* similar to the proposed one. In our approach, we do not intend to characterise the cases in terms of problem and solution parts: our Retrieve process is not focused on finding which songs in the Case Base are the most *similar* to the input song. Instead, we look for useful cases [6], that is, previous users' playlists that are useful to achieve the requested goals. The idea is to retrieve a subset of those playlists that include s **(G1)**, are varied **(G2)**, and contain relevant patterns, or else are coherently ordered **(G3)**.

Before presenting the Retrieve process, we introduce the *Relevance function* and the *Rating function*. The Relevance function $Rel(q, t)$, defined in Sect. 5.1, takes as input a sequence of songs q, and a song t contained in q, and returns the degree in which q is a *relevant pattern* for t. The Rating function $\rho(p, s)$, defined in Sect. 5.2, takes in input a playlist $p \in \mathcal{C}$ and the chosen song s, and returns a combined measure of the variety of p and the relevant patterns in p that contain s. At the end of the section, we present the Retrieve process that, given all the playlists of $\mathcal{C}$, returns only the k playlists that are more useful to fulfil our goals.

5.1 Patterns and Relevance

Let q be a sequence of songs. We say q a *pattern* if q is sub-sequence of at least one playlist of $\mathcal{C}$, where $q = (t_1, t_2, .., t_m)$ is *sub-sequence* of $p = (s_1, s_2, .., s_n)$ when the following condition holds: $Sub(q, p) = \exists i : \forall j, 1 \leqslant j \leqslant m, t_j = s_{i+j}$.

The *pattern count* $\phi(q) = |\{p \in \mathcal{C} : Sub(q,p)\}|$ is the number of playlists of $\mathcal{C}$ that q is sub-sequence of.

Let t be a song, $Q(t) = \{q : \Lambda(q) \leqslant \theta \wedge In(t,q) \wedge \exists p \in \mathcal{C} : Sub(q,p)\}$ be the set of patterns, with at most θ songs, that contain t, and $q \in Q(t)$ be one of these patterns. To assess whether q is a relevant pattern for t, we can consider how many times t occurs in the playlists of $\mathcal{C}$ together with the other songs of q. According to co-occurrences analysis, in fact, the higher this number, the higher the evidence that q is a relevant pattern for t. Thus, it makes sense to use the pattern count $\phi(q)$ as a measure of the relevance of q for t. Nevertheless, the actual value of $\phi(q)$ can be biased by two properties of q: its length and the popularity of its songs. In general, short patterns or patterns made of very popular songs have a higher probability of occurring in many playlists, independently from their relevance for a specific song.

Example 1. To understand why shorter patterns are likely to have a higher pattern count, consider the patterns $(S2, S3)$ and $(S2, S3, S4)$ in Table 1. Notice that $(S2, S3)$ is shorter than $(S2, S3, S4)$, and $\phi(\,(S2, S3)\,) > \phi(\,(S2, S3, S4)\,)$. This is not surprising, for $(S2, S3)$ is contained in $(S2, S3, S4)$, so it occurs at least every time that $(S2, S3, S4)$ occurs in some playlist. To understand why patterns made of popular songs are likely to have a higher pattern count, consider the patterns $(S2, S3)$ and $(S9, S2)$, both containing $S2$. In the first case, $S2$ is followed by a "popular" song ($S3$ occurs 6 times in Table 1); in the second case, $S2$ is preceded by a less popular song ($S9$ occurs only 3 times). Notice that $\phi(\,(S2, S3)\,) > \phi(\,(S9, S2)\,)$; this is not surprising, for $S3$ has a higher probability than $S9$ to co-occur with $S2$, due to its high popularity.

Table 1. A sample repository and the list of patterns with $\phi > 1$.

Playlists	Patterns	
$A = (\underline{S1, S2}, S4)$	$(S1, S2)$	$(\phi = 2)$
$B = (\underline{S1, S2, S3, S4}, S5, S7, S1, S8)$	$(S2, S3, S4)$	$(\phi = 3)$
$C = (\underline{S4, S6, S9, S2}, \underline{S2, S3})$	$(S2, S3)$	$(\phi = 6)$
$D = (\underline{S2, S3}, \underline{S2, S3, S4})$	$(S3, S4)$	$(\phi = 3)$
$E = (\underline{S2, S3}, \underline{S4, S6}, S5)$	$(S4, S6)$	$(\phi = 2)$
$F = (\underline{S9, S2, S3}, S5, S9, S1)$	$(S9, S2)$	$(\phi = 2)$

In a nutshell, short patterns and patterns made of popular songs are likely to have a high pattern count, independently from being relevant to a particular song. Therefore, to estimate the relevance of a pattern q to a song t, we combine $\phi(q)$ with two factors that balance the effect of the length of q, and of the popularity of its songs. We define the *Relevance* of a pattern q for a song t as follows:

$$Rel(q,t) = \phi(q) \cdot \frac{\alpha^{\theta - \Lambda(q)}}{\psi^{\beta}(q,t)}$$

where $\alpha^{\theta-\Lambda(q)}$ and $\psi^{\beta}(q,s)$ are the two factors intended to reduce the bias of the length of q, and of the popularity of its songs, on the relevance value. Hereafter, we explain in detail these two factors.

Length factor. Let $q_1 \in Q(t)$ and $q_2 \in Q(t)$ be two patterns, with same pattern count and different length (e.g. $\Lambda(q_1) > \Lambda(q_2)$). In this case, we consider q_1 more relevant for t than q_2, because q_1 occurs the same number of times than the (more probable) shorter pattern q_2. Moreover, since q_1 is longer, it contains more songs co-occurrent with t than q_2. For this reason, we introduce a *length factor* $\alpha^{\theta-\Lambda(q)}$ that we multiply $\phi(q)$ for, in order to substantially decrease the relevance of shorter patterns. The shorter q (that is, the more distant is $\Lambda(q)$ to the maximum pattern length θ), the smaller this factor, the smaller the relevance value. The parameter α can be tuned inside $(0, 1]$, to control if the length of q should affect more (small α) or less (high α) the relevance value.

Popularity factor. Let $q_1 = (t, u)$ and $q_2 = (t, v)$ be two patterns of two songs with the same pattern count (e.g. $\phi(q_1) = \phi(q_2) = 5$), and let u be more popular than v (e.g. u occurs in 600 playlists of $\mathcal{C}$, while v only in 6). Notice that u occurs relatively few times in $\mathcal{C}$ preceded by t (5 out of 600), while v occurs relatively many times preceded by t (5 out of 6). In the whole, q_2 appears in $\mathcal{C}$ more frequently *in relation to the popularity of its songs*; in this sense it is more relevant for t. For this reason, we introduce a *popularity factor* $\psi^{\beta}(q, t)$ that we divide $\phi(q)$ by, in order to decrease the relevance of patterns made of popular songs. This factor acts as a measure of inverse songs' frequency of q; precisely, $\psi(q, t)$ is the number of times that the sequence q occurs in $\mathcal{C}$ "independently from t": $\psi(q, t) = |\{p \in \mathcal{C} : \exists i : \forall j, 1 \leqslant j \leqslant m, q_j = p_{i+j} \vee q_j = t\}|$. The parameter β can be tuned inside $[0, 1]$ to control if the popularity of the songs should affect more (high β) or less (small β) the relevance value.

Example 2. To understand the effects of α and β on the relevance value, consider the patterns in Table 1, ranked according to how *relevant* they are to a specific song, $S3$. Table 2 shows the relevance values, using different values of α and β.
Notice that $(S2, S3)$ has the highest pattern count, and indeed is the most relevant pattern when $\alpha = 1$ and $\beta = 0.5$. However, if we decrease the relevance of *short patterns* ($\alpha = 0.5$), then the most relevant pattern becomes $(S2, S3, S4)$, that occurs less times but is longer. Similarly, if we decrease the effect of popular songs ($\beta = 1$), then the most relevant pattern is $(S2, S3, S4)$, for it occurs 3 out of the 3 times that $S2$ and $S4$ occur in some playlist of Table 1 separated by one song.

Table 2. Relevance to $S3$ of the patterns from Table 1

Patterns $q_i \in Q(S3)$	$\Lambda(q_i)$	$\phi(q_i)$	$\psi(q_i, S3)$	$Rel(q_i, S3)$ ($\alpha = 1$ $\beta = 0.5$)	$Rel(q_i, S3)$ ($\alpha = 0.5$ $\beta = 0.5$)	$Rel(q_i, S3)$ ($\alpha = 1$, $\beta = 1$)
$(S2, \underline{S3}, S4)$	3	3	3	1.73	**1.73**	**1**
$(S2, \underline{S3})$	2	6	8	**2.12**	1.06	0.75
$(\underline{S3}, S4)$	2	3	4	1.50	0.75	0.75

5.2 Coherence and Retrieval

We are now able to define the Rating function $\rho(p, s)$, that assigns a high value to playlists that are varied and coherent with the input song s. This function is a combination of $Var(p)$ (the variety of the playlist) and $Coh(p, s)$ (the coherence of the playlist to the input song), where $Coh(p, s)$ is defined as the sum of the relevance values of the patterns, present in p, that contain s:

$$Coh(p, s) = \sum_{q \in \Omega(s,p)} Rel(q, s)$$

where $\Omega(s, p) = \{q : \Lambda(q) \leqslant \theta \wedge In(s, q) \wedge Sub(q, p)\}$. The more the relevant patterns that contain s in p, and the higher their relevance to s, the higher $Coh(p, s)$. The Rating function $\rho(p, s)$ is defined as a combination of variety and coherence:

$$\rho(p, s) = Var(p) \cdot Coh(p, s), \quad \forall p \in \mathcal{C}$$

As a result, a playlist $p \in \mathcal{C}$ has a high value of $\rho(p, s)$ if it contains s, is varied, and is coherently ordered. Actually, this is the kind of playlist that we intend to retrieve from $\mathcal{C}$. Let k be the number of playlists to retrieve, the Retrieve process returns the first k playlists of $\mathcal{C}$, every playlist $p \in \mathcal{C}$ ranked according to $\rho(p, s)$.

6 Reuse

The goal of the Reuse process is to transform the k retrieved playlists into a single, recommended playlist of a specific length λ. Notice that the retrieved playlists can have any length, so they cannot be directly returned as recommendations. The approach we take for Reuse is that of Constructive Adaptation [19]. Following this approach, we use the elements of the retrieved cases (i.e. the songs from the retrieved playlists) to build a new combination of elements (i.e. a new playlist), that is a solution for the current problem. Building this playlist is seen as a search process, guided by the information contained in the Case Base (in the form of relevant patterns).

Basically, Constructive Adaptation [19] is composed of two processes: *Hypotheses Generation* and *Hypotheses Ordering*. Hypotheses Generation determines how partial solutions are extended in the search process (i.e. which songs from the retrieved playlists can be combined and how). Hypotheses Ordering uses the knowledge from the Case Base to rank the nodes visited during the search (i.e. the sequences of songs generated so far) in order for the search process to follow the best paths to the solution. Thus, the Reuse process is a search process where:

- the goal is to find a playlist of λ songs, that includes the input song s;
- every node in the search is a sequence made of songs from the retrieved playlists;
- the singleton sequence (s) is taken as the initial node;

– the successors of every node contain the same sequence of songs of that node, plus another song, either at the beginning or at the end of the sequence.

The fastest method to reach a sequence of length λ starting from (s) (a sequence of length 1) is to always explore the search tree "as deep as possible", that is, to always visit first the nodes with the longest sequences of songs found so far. For this reason, the system uses a Depth-First Search process: at first, it expands the initial node (s) by generating all its successors; at each subsequent step, it considers one of the most recently generated nodes, generates its successors, and ranks them according to a heuristic; if at any moment it finds a node with λ songs, returns it as the solution; if otherwise it finds a node with no successors, it backtracks to expand another of the most recently generated nodes.

6.1 Hypotheses Generation

The nodes of the search are sequences made of elements from the retrieved cases, that is, of songs from the retrieved playlists. Let $T = (t_1, t_2, .., t_n)$ be a node, and let T' be a successor of T. T' contains the same songs of T, and another song u from the retrieved playlists, either at the beginning or at the end of T'. We distinguish these two cases with the notations $\langle u + T \rangle$ and $\langle T + u \rangle$. Let, for instance, be $T' = \langle u + T \rangle$. Notice that in T', u occurs before t_1. In order for T' to be coherently ordered, u should occur before t_1 at least in one retrieved playlist; basically we consider T' a valid successor of T if $\exists p \in Ret(s) : Sub((u, t_1), p)$, where $Ret(s)$ is the set of k retrieved playlists. Similarly, a sequence $\langle T + u \rangle$ is a valid successor of T if $\exists p \in Ret(s) : Sub((t_n, u), p)$. The set of all the valid successors of a node T is defined as $Succ(T) = \{\langle u + T \rangle, \forall u \in Pre(T) \vee \langle T + u \rangle, \forall u \in Post(T)\}$, where $Pre(T) = \{u : \exists p \in Ret(s) : Sub((u, t_1), p)\}$ and $Post(T) = \{u : \exists p \in Ret(s) : Sub((t_n, u), p)\}$.

In the next subsection, we will describe how such sets of successors are sorted in the search process in order to finally reach a sequence of λ songs that is varied, coherently ordered, and relevant to s.

6.2 Hypotheses Ordering

In ordering the successors of a node, we promote those sequences that are varied and include relevant patterns. In other words, given any successor T' of T (where u is the added song), we determine if T' is a *good successor* of T using the heuristic function $H(T') = Rel(T', u) \cdot Var(T')$. Using H, we are able to rank highly a node T' that contains a varied and coherently ordered sequence of songs. However we cannot determine whether the successors of T' will be varied and coherently ordered sequences as well. To solve this issue, we introduce in the heuristic a parameter L, called look-ahead factor, such that:

– If $L = 0$, we evaluate H on T', as above;
– If $L = 1$, we evaluate H on *all the successors* of T'; then we return the maximum value found as the heuristic value for T';

– If $L = 2$, we evaluate H on *all the successors at depth 2* of T', that is, the successors of the successors of T'; then we return the maximum value found as the heuristic value for T'; and so on.

The higher is L, the more reliable is the heuristic; on the other hand, the number of requested evaluations of H grows steeply with L. The value of L is usually a compromise between a good heuristic and a reasonable performance. Finally, we define the heuristic function as follows:

Definition 2. ***(Look-ahead heuristic)*** *Let T be a node of the search tree; any successor T' of T is ranked using the heuristic function:*

$$H'(T', L) = \begin{cases} H(T') & \text{if } L = 0 \\ \max_{T'' \in Succ(T')} H'(T'', L-1) & \text{if } L > 0 \end{cases}$$

Summing up, the Reuse process follows a Constructive Adaptation approach in which a) Hypotheses Generation explores the possible combinations of the songs of the retrieved playlists and b) Hypothesis Ordering ranks the partial sequences according to a heuristic based on relevance and variety. At the end, the result is a playlist of length λ that combines the best sub-sequences in the retrieved playlists into a varied and coherent recommendation.

Example 3. Let $S3$ be the input song, and $\lambda = 5$ the desired length. Let the repository of Table 1 be the Case Base, and $Ret(S3) = \{B, C, E, F\}$ be the subset of $k = 4$ playlists retrieved. Consider how, in the Reuse process, these playlists are combined using a search process (for brevity, the computation of H' is omitted):

1) Generate the successors of $(S3)$ and sort them using H': $(\underline{S3}, S4)$, $(S2, \underline{S3})$, $(\underline{S3}, S5)$. Next node to visit: $(S3, S4)$.

2) Generate the successors of $(S3, S4)$ and sort them using H': $(\underline{S3, S4}, S6)$, $(S2, \underline{S3, S4})$, $(\underline{S3, S4}, S5)$. Next node to visit: $(S3, S4, S6)$.

3) Generate the successors of $(S3, S4, S6)$ and sort them using H': $(S2, \underline{S3, S4, S6})$, $(\underline{S3, S4, S6}, S9)$, $(\underline{S3, S4, S6}, S5)$. Next node to visit: $(S2, S3, S4, S6)$.

4) Generate the successors of $(S2, S3, S4, S6)$ and sort them using H': $(S9, \underline{S2, S3, S4, S6})$, $(S1, \underline{S2, S3, S4, S6})$, $(\underline{S2, S3, S4, S6}, S9)$, $(\underline{S2, S3, S4, S6}, S5)$. Next node to visit: $(S9, S2, S3, S4, S6)$.

5) The node $(S9, S2, S3, S4, S6)$ has $\lambda = 5$ songs, hence is returned as the solution.

Notice that the solution is a varied and coherent combination of sub-sequences from the retrieved playlists: $(S2, S3, S4)$ from B, $(S4, S6)$ and $(S2, S3)$ from C, $(S2, S3, S4, S6)$ from E and $(S9, S2, S3)$ from F.

7 Experimental Results

We have tested the described CBR recommending system using a collection of playlists from MusicStrands[1]. In the Case Base, we have represented playlists

[1] For more information visit: http://www.musicstrands.com.

as pairs $(track, artist)$, being *track* and *artist* two integers that allow us to univocally identify every track and every artist in the MusicStrands database. We have pre-processed the collection of playlists, discarding those with less than 5 songs, with more than 50 songs, and those alphabetically ordered (backwards and forwards) along the names of the track and/or of the artist. Eventually, we have assembled the Case Base using a subset (about 300,000) of playlists from MusicStrands.

Concerning parameters, we were interested in recommending playlists where tracks were mostly never repeated ($\gamma_{\text{track}} = 200$) and artists were hardly repeated ($\gamma_{\text{artist}} = 50$). We have chosen $\theta = 5$ as the maximum pattern length (longer patterns very rarely appeared more than once), $k = 50$ as the number of retrieved playlists, $\alpha = 0.5$ and $\beta = 1$ as the length factor and popularity factor for the relevance, and $L = 1$ as the look-ahead factor.

Table 3 shows six sample recommendations, where the desired length is $\lambda = 10$, and the input songs have been chosen from different genres and periods. Notice that the suggested songs are similar (both acoustically and in terms of metadata) to the input songs, especially those that appear closely before or after them in the proposed sequences[2]. On the whole, the songs in the playlists sound well in the presented order, and their genres are never too distant from the one of the input song.

It is also worth noticing that many of the proposed songs are not very popular, yet they form a coherently ordered playlist with the input songs. This is due to effect of the popularity factor in the relevance function ($\beta = 1$), by which we dampen playlists made of popular songs. In this way, we enable the user to *discover new music*, because we are suggesting songs the user has probably never listened to, and yet will appreciate, since they are correlated with the input song. Should users long for recommendations with more popular songs, they would just have to lower this value. For instance, in Table 4 we present two recommendations, obtained with the same input songs of Test 2 and Test 4, but generated with $\beta = 0$, that is, *without* dampening popular songs. Consider, for instance, the song *American Idiot*, recommended in Test 2b ($\beta = 0$) before the input song *American Pie*. These two songs co-occur in sequence 7 times in the Case Base, which is, in general, a relevant pattern count. However, this pattern count becomes irrelevant when compared to the popularity of *American Idiot*, which is a very popular song (it appears in 8,479 playlists) and *only 7 out of 8,479 times* it is followed by *American Pie*. As a result, the system does not recommend this sequence in Test 2 ($\beta = 1$), where pattern counts are dampened for sequences made of popular songs.

As a matter of fact, by proposing different recommendations from the same input, the parameter β allows to improve the *diversity* of the solutions. This is indeed an important feature for a recommending system, and a key issue for CBR systems [5].

[2] A 30" excerpt of every song, along with detailed metadata, is available on MusicStrands web-site.

Table 3. Sample of recommended playlists

Test 1. Input song: **Only Shallow** (*My Bloody Valentine*)	Test 2. Input song: **American Pie** (*Don McLean*)
Kool Thing *Sonic Youth*	We're An American Band *VV.AA.*
Stupid Preoccupations *Vic Chesnutt*	Sweet Home Alabama *Lynyrd Skynyrd*
Seed Toss *Superchunk*	More Than a Feeling *Boston*
Flyin' The Flannel *Firehose*	Bad Moon Rising
Only Shallow *My Bloody Valentine*	*Creedence Clearwater Revival*
Exhibit A *The Features*	**American Pie** *Don McLean*
Lover's Spit *Broken Social Scene*	Mr. Blue Sky *Electric Light Orchestra*
Gigantic *The Pixies*	Switch *Will Smith*
Kamera *Wilco*	This Love *Maroon 5*
From Blown Speakers	Walkie Talkie Man *Steriogram*
The New Pornographers	Walkin' On The Sun *Smash Mouth*

Test 3. Input song: **Mrs. Robinson** (*Simon & Garfunkel*)	Test 4. Input song: **Soldier** (*Destiny's Child*)
Unchained Melody	Let Me Love You *Mario*
The Righteous Brothers	Hush *LL Cool J*
Kicks *Paul Revere & The Raiders*	Red Carpet (Pause, Flash)
Cherry Hill Park *Billy Joe Royal*	*R. Kelly*
Windy *The Association*	Hot 2 Nite *New Edition*
Sunshine Superman *Donovan*	Wonderful *Ja Rule*
Mrs. Robinson *Simon & Garfunkel*	My Prerogative *Britney Spears*
Nights In White Satin *The Moody Blues*	Two Step *Ciara*
Only One *Yellowcard*	**Soldier** *Destiny's Child*
Pain *Jimmy Eat World*	Only U *Ashanti*
Quiet Ones *Tears For Fears*	Pass Out *Ludacris*

Test 5. Input song: **Roots Bloody Roots** (*Sepultura*)	Test 6. Input song: **Strangers In The Night** (*F. Sinatra*)
Satan Spawn, The Caco-Daemon	It Had To Be You *Steve Tyrell*
Deicide	Jamaica Farewell *Desmond Dekker*
Rapture *Morbid Angel*	Just The Way You Are *Diana Krall*
Serenity In Fire *Kataklysm*	Let's Fall In Love *Diana Krall*
Roots Bloody Roots *Sepultura*	Nunca Es Para Siempre
Plague Rages *Napalm Death*	*Presuntos Implicados*
Blend As Well *Coalesce*	**Strangers In The Night** *F. Sinatra*
God Send Death *Slayer*	Candy Man, The *Sammy Davis Jr.*
Token *Damad*	Unforgettable *Nat King Cole*
Heaven In Her Arms *Converge*	What A Wonderful World
Fear Of Napalm	*Louis Armstrong*
Terrorizer	Falling In Love Again *Billie Holiday*

Table 4. Sample of recommended playlists, without dampening popular songs

Test 2b. Input song: **American Pie** (*Don McLean*)	Test 4b. Input song: **Soldier** (*Destiny's Child*)
Behind These Hazel Eyes *Kelly Clarkson*	Disco Inferno *50 Cent*
Beverly Hills *Weezer*	Mockingbird *Eminem*
I Just Wanna Live *Good Charlotte*	Obsession *Frankie J*
American Idiot *Green Day*	I Just Wanna Live *Good Charlotte*
American Pie *Don McLean*	Boulevard Of Broken Dreams *Green Day*
Hotel California *The Eagles*	Since U Been Gone *Kelly Clarkson*
Cocaine *Eric Clapton*	Two Step *Ciara*
Emerald Eyes *Fleetwood Mac*	**Soldier** *Destiny's Child*
Carry On Wayward Son *Kansas*	Drop It Like It's Hot *Snoop Dogg*
Sweet Home Alabama *Lynyrd Skynyrd*	Get Back *Ludacris*

8 Conclusions and Future Work

We have presented a CBR system for recommending playlists made of meaningful sequences of songs. The Case Base is a collection of playlists of different quality, provided by different people. While most recommendation systems are concerned with suggesting an unorganised set of songs, "similar" to those selected by the user, our work focuses on recommending meaningful sequences of songs (playlists). The proposed recommendations are not deeply "personalised", since the model we have of the user requesting a playlist is given only by the selected (input) song. Future works will explore how to further constrain the playlist to recommend by acquiring a larger input from the user.

This paper contributes to develop Case-based Reasoning in a variety of ways. First, we address the important issue of reasoning about sequential cases (here, in the form of coherently ordered playlists). Both the Case Base and the solution provided by the CBR system are based on the sequential ordering of songs; the problem-solving process of the CBR system exploits the informational content of the Case Base to provide a solution with a coherent order.

A second issue of interest is that our Case Base contains only *solutions* and not *problems*. In the usual CBR approach, the Case Base contains pairs (p, s) (p is a problem, s is a solution); to find the solution for a new problem p', the system retrieves from the Case Base a set of *similar problems*, takes into account their solutions, and returns one of them, or a combination of them, as the solution for p'. In our approach, instead, the Case Base contains only *solutions* (playlists), and no explicit *problems*: we know that the playlists in the Case Base are meaningful, but we ignore for which input songs they are good recommendations. In this context, we have designed the simplest CBR approach that can exploit the information contained in a large Case Base of meaningful sequences. We have explicitly avoided complex user models, which are a common feature of Collaborative Filtering systems. User models are not helpful in relation

to our goal: they seek relations between people and preferred songs or playlists, but disregard any sequential information *contained* in the playlists.

A third issue of interest is the Reuse process: rather than recommending an existing playlist, we combine the retrieved playlists using the same criteria (variety and coherence) that we used in the Retrieve process. In this way we recommend a playlist that has the desired length λ and higher variety and coherence values than those of the retrieved playlists.

Further, we have investigated the issue of applying co-occurrences analysis in a musical context. In this approach, we have opted for a minimal *Problem Description* (only one song and the length of the desired recommendation) in order to generate meaningful playlists by exploiting the co-occurrences present in the Case Base. We have avoided more constrained inputs (e.g. two songs, such that the recommendation starts with the first one and ends with the other, and is coherently ordered). The problem with over-constrained input is that the system returns low quality solutions when the constraints are too strong (e.g. a user requests a playlist to start with a rap song and to end with a requiem). In the future, however, we plan to enlarge the input capabilities of the user, to partially constrain the recommended playlist. For instance, the user could provide a short sequence of songs, and the system could recommend a playlist with *most* of them; or else the user could indicate a specific genre, or tag, to consider or to avoid in the recommended playlist.

We have tested the system using a collection of manually-generated playlists, but in the future we plan to work with other forms of sequential music data. For instance, MusicStrands provides MyStrands, an iTunes plug-in that recommends new songs to the user, on the basis of the last songs effectively played in iTunes. We plan to improve this tool with the introduction of an explicit notion of sequentiality, in order to recommend *on-line, meaningful, personalised playlists*, on the basis of previous sequences of songs effectively played by the user.

The evaluation of the *quality* of the recommendation is an open issue for recommender systems. We agree with Hayes et al. [9] that user satisfaction can only be measured in an on-line context. Although the proposed system has not yet been evaluated by a significant number of users, we have deployed it on the Internet[3]. Users can select an input song, the desired length, and other parameters, and can compare the (meaningfully ordered) recommendation generated with the presented approach with the playlist proposed by the MusicStrands recommender. We plan to employ the users' feedback as an evaluation for our work and, once recommendations have been positively *revised* by the users, to *retain* in the MusicStrands repository those that have been approved as good playlists, to use them as new cases for future recommendations.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–

[3] http://labs.musicstrands.com

59, 1994.
2. M. Alghoniemy and A.H. Tewfik. User-defined music sequence retrieval. In *Proc. ACM Multimedia*, 356–358, 2000.
3. J.-J. Aucouturier and F. Pachet. Scaling up Music Playlist Generation. In *Proc. of the 3rd IEEE Intl. Conf. on Multimedia and Expo*, 2002.
4. P. Avesani, P. Massa, M. Nori, and A. Susi. Collaborative radio community. In *Proc. of Adaptive Hypermedia*, 2002.
5. K. Bradley and B. Smyth. Improving recommendation diversity. In *Proc. of the 12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001.
6. H.-D. Burkhard. Extending some Concepts of CBR – Foundations of Case Retrieval Nets. In *Case-Based Reasoning Technology – From Foundations to Applications*, 9:17–50, 1998.
7. D. B. Hauver and J. C. French. Flycasting: Using Collaborative Filtering to Generate a Playlist for Online Radio. In *Proc. of the Intl. Conf. on Web Delivering of Music*, 2001.
8. C. Hayes and P. Cunningham. Smart radio: Building music radio on the fly. In *Expert Systems*, 2000.
9. C. Hayes, P. Massa, P. Avesani and P. Cunningham. An online evaluation framework for recommender systems. In *Proc. of the RPEC Conference*, 2002.
10. C. Hayes and P. Cunningham. Context-boosting collaborative recommendations. *Knowledge-Based Systems*, 17:131-138, 2004.
11. T. Hofmann and J. Puzicha. Statistical models for co-occurrence data. *Memorandum, MIT Artificial Intelligence Laboratory*, 1998.
12. B. Logan. Music recommendation from song sets. In *Proc. of the 5th ISMIR Conference*, 2004.
13. C. Manning and H. Schütze. Foundations of Natural Language Processing. 1999.
14. R. López de Mántaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M-L. Maher, M.T. Cox, K. Forbus, M. Keane, A. Aamodt, I. Watson. Retrieval, reuse, revision, and retention in case-based reasoning. *Knowledge Engineering Review*. In press, 2006.
15. F. Pachet, G. Westerman, and D. Laigre. Musical data mining for electronic music distribution. In *Proc. of the Intl. Conf. on Web Delivering of Music*, 2001.
16. E. Pampalk, T. Pohle, and G. Widmer. Dynamic Playlist Generation Based on Skipping Behaviour. *Proc. of the 6th ISMIR Conference*, 2005.
17. S. Pauws and B. Eggen. PATS: Realization and User Evaluation of an Automatic Playlist Generator. In *Proc. of the Intl. Conf. on Music Information Retrieval*, 2002.
18. J. Platt, C. Burges, S. Swenson, C. Weare, and A. Zheng. Learning a gaussian process prior for automatically generating music playlists. In *Advances in Neural Information Processing Systems*, 14:1425-1432, 2002.
19. E. Plaza and J.-L. Arcos. Constructive adaptation. *Lecture Notes in Artificial Intelligence*, 2416:306–320, 2002.
20. T. Pohle, E. Pampalk, and G. Widmer. Generating similarity-based playlists using traveling salesman algorithms. In *Proc. of the Intl. Conf. on Digital Audio Effects*, 2005.
21. R. Ragno, C. J. C. Burges, and C. Herley. Inferring Similarity Between Music Objects with Application to Playlist Generation. *ACM Multimedia Information Retrieval*, 2005.
22. K. Wang. Discovering Patterns from Large and Dynamic Sequential Data. *Journal of Intelligent Information System*, 8–33, 1995.