A Combinatorial Pattern Generation Approach to Content-Based Music Selection

François **Pachet**
SONY CSL Paris, 6, rue Amyot 75005, Paris, FRANCE
Tel: (33) 1 44 08 05 16, Fax: (33) 1 45 87 87 50, E-mail: pachet@csl.sony.fr

Abstract

We describe a system to build on-the-fly *music programs* that satisfy user preferences, while ensuring global musical consistency. Each item in a database is described by various musical attributes, such as style or tempo. Music programs are seen as sequences of items satisfying predefined constraints. Building sequences is seen as a combinatorial pattern generation problem. We introduce three types of constraints to specify such sequences: constraints of similarity, difference and cardinality, with efficient solving algorithms for each of them.

1. Content-Based Music Selection

There is a growing need today for intelligent interfaces to access music catalogues. First, high-speed networking enables large scale access to huge multimedia databases. Second, record companies & broadcasters want a more flexible exploitation of their catalogues. Finally, users want to compile their own selections on a portable format, according to their personal taste. Although it is difficult to characterize user preferences, there are two contradictory parameters for satisfying user's musical tastes: *1)* users want to find titles they already know, *2)* users want to discover *new music*. Of course finding the right compromise is the key issue.

However, work on music delivery has so far concentrated on networks problems, and not on end user interfaces. Existing interfaces exploit either *statistical* information from user actions (e.g. [1]), or strictly marketing-driven strategies such as [2]. In all cases, the problem of music catalogue access is seen as a purely database/network problem. The result is a poor exploitation of the catalogue, and poor user satisfaction in music access.

Instead, we propose a technique for proposing users *coherent sequences of music titles*, rather than collections of individual titles. We produce such sequences by considering the problem as a combinatorial pattern generation problem.

2. Recital Composer

The idea in RecitalComposer is to exploit intelligently a database of music titles, by producing sequences of titles satisfying explicit generic properties. The properties are of two kinds: 1) explicit user preferences, and 2) properties on sequences. User preferences are expressed at any level of detail: either a preferred musical style (e.g. "Jazz"), a given song or set of songs, a given author, voice type, etc.

Properties of sequences are expressed as constraints, as explained below. Finally, a constraint solver finds all the solutions of the constraint problem in a reasonable time thanks to specialised constraint propagation algorithms.

2.1 A Database of Music Titles

A database of music titles is built, in which each music title is described by musical attributes. The attributes necessary to build interesting sequences are of two sorts: 1) administrative attributes such as name of title, author, duration, 2) musical attributes such as: musical style, type of voice, type of instrumentation, instruments used, type of melody. Each attribute takes its value within a predefined taxonomy designed with experts from Sony Music France. For instance, styles are picked out of a *taxonomy of styles*. The taxonomy contains 150 styles, and includes a relation of similarity between styles. For instance, "Jazz-Blues" is *close* to "Jazz-Crooner", but not to "Classical-dodecaphonic".

3. Constraints on Music Programs

Instead of allowing a fully general constraint language, which leads to difficult to use and inefficient algorithms, we identified three main classes of constraints to specify sequences: 1) constraints expressing *similarities* in the sequence, which will ensure some sort of continuity and coherence, 2) constraints expressing *differences* in the sequence, which will bring novelty and surprise in the result, and 3) constraints expressing *cardinalities*, i.e. numbers of items satisfying given properties, expressing explicit user preferences. The combination of these constraints creates a complex combinatorial problem (especially if the database contains about 1 million titles) which is solved by an appropriate constraint solver.

3.1 Constraints of Similarity

This constraint allows to state that within a given range, the items are successively similar to each other. The similarity is defined by a binary predicate holding on one given attribute *j*. The general formulation is :

S(a, b, j, similar(,)) meaning that :

For every item p_i, $i \in [a, b-1]$, $similar(p_i.a_j, p_{i+1}.a_j)$ is true.

Where a and b are integers representing indexes, j is an attribute, and *close(,)* is a two variable predicate. Each of the variable of the predicate denotes an item's *jth* attribute. For instance, this constraint allows to state that all pieces in a given contiguous range (say the first 10) should have "close" styles, where closeness is the similarity relation of the underlying style classification.

3.2 Constraints of Difference

This constraint allows to state difference of attributes on a set of contiguous items. Its general formulation is :

D(I, j) meaning that:

All items p_i, $i \in I$, have pairwise different values for attribute j. Where I is a set of item indexes, j is an attribute index. For instance, this constraint class allows to state that all pieces in a given range (say the first 10) should have different authors, or different styles, etc.

3.3 Constraints of Cardinality

These constraints allow to impose properties on *sets of items*. They are the most difficult from a combinatorial point of view, because they state properties on the *whole* sequence. There are two such cardinality constraints.

3.3.1 Cardinality on items

This constraint allows to state that the number of items whose attribute j belongs to a given set E is within [a, b]. The general formulation is :

CI(I, j, a, b, E) meaning that $| \{i \in I; p_i.a_j \in E \} | \in [a, b]$

Where I is a set of item indexes, j is an attribute index, a and b are integers and E is a subset of the possible values of attribute j. For instance, this constraint can be used to state that the number of pieces within a given range (e.g. the first 10 pieces), whose style is "Rock", should be comprised between, say, 4 and 6.

3.3.2 Cardinality on attribute values

This constraint class allows to state that the number of different values for attribute j of a number of items is within [a, b]. The general formulation is:

CA(I, j, a, b) meaning that: $| \{p_i.a_j \; i \in I\} | \in [a, b]$

Where I is a set of item indexes, j is an attribute index, a and b are integers. This constraint can be used to state that three pieces should have at least two different tempo.

3.4 Example

The constraint algorithm is based on a forward-checking loop [3], increased with specialized filtering methods for each of the constraints. It finds all the solutions of the constraint problem in a reasonable time, and was validated by a few examples of realistic music programs. For instance, a typical music program is the following:

- Contains 12 different titles (to fit on a CD / Minidisc).
- Path is continuous stylistically: each piece belongs to a style “close” to the style of the preceding piece.
- Starts by a “Soul-Jazz” piece, and ends by a “Soul-Crooner” piece.
- Starts rather slowly and end quickly.
- Evolves continuously tempo-wise: each piece has a tempo which is close to the tempo of the preceding piece.
- All authors are different.

This program is represented by the following constraints :

- Cardinality constraint on first and last piece to set imposed styles,
- Similarity constraint on styles,
- Cardinality constraint on tempo for first and last piece.
- Similarity constraint on tempos.
- Difference constraint on pieces (index attribute)
- Global difference constraint on authors

3.5 The interface

The current interface is designed for professional use, and allows to specify constraint sets, run them on an arbitrary music catalogue (see Figure 1), and visualize the result. A specialized interface allows to specify for each constraint all the required parameters (see Figure 2).

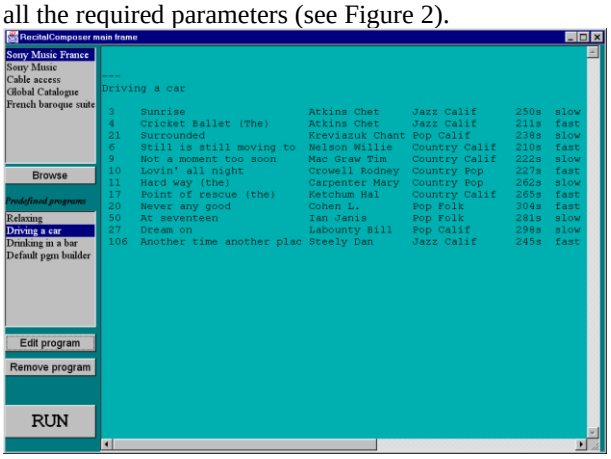

Figure 1. The Interface for building music sequences.

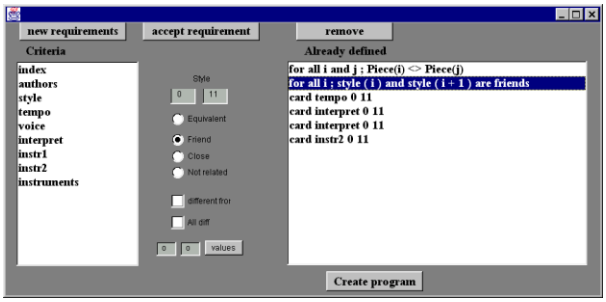

Figure 2. The interface for specifying constraint sets.

4. Conclusion

We claim that the combinatorial pattern generation approach is a good solution to the problem of intelligent access to music catalogues. This approach requires sophisticated combinatorial search methods, together with efficient specialized constraint classes, that we designed successfully. Such an approach should allow both record companies to better exploit their catalogue and user to listen to music more adapted to their tastes and desires of novelty, because it creates music sequences which satisfy user preferences, while providing them with novel music.

5. References

[1] Amazon Music Store: http://www.amazon.com
[2] Firefly system: http://www.firefly.com
[3] Roy, P. Pachet, F. Reifying Constraint Satisfaction in Smalltalk. *Journal of Object-Oriented Programming* (JOOP), 10 (4), pp. 43-51, July/August 1997.