SCALING UP MUSIC PLAYLIST GENERATION

Jean-Julien Aucouturier, Francois Pachet

SONY Computer Science Laboratory
6, rue Amyot, 75005 Paris, France
Email: {jj,pachet}@csl.sony.fr

ABSTRACT

The issue of generating automatically sequences of music titles that satisfy arbitrary criteria such as user preferences has gained interest recently, because of the numerous applications in the field of Electronic Music Distribution. All the approaches proposed so far suffer from two main drawbacks: reduced expressiveness and incapacity to handle large music catalogues. We present in this paper a system that is able to produce automatically music playlists out of large, real catalogues (up to 200,000 titles), and that can handle arbitrarily complex criteria. We describe the basic algorithm and its adaptation to playlist generation, and report on experiments performed in the context of the European project Cuidado.

1. INTRODUCTION: THE COMBINATORIAL APPROACH TO PLAYLIST GENERATION

The goal of Electronic Music Distribution is to make accessible large catalogues of music titles to the largest possible audience. Most proposals of EMD systems so far have followed a purely database-oriented approach, in that users are proposed individual titles, either queried explicitly (e.g. e-compil.fr, PressPlay.com) or through recommendation systems (e.g. Amazon). Departing from these approaches, we have introduced in [1] the idea of producing automatically sequences of music titles - playlists, instead of individual titles. These sequences are produced automatically from a set of so-called global constraints, which specify properties of the whole sequence, such as:

- ”All Different” constraint: the playlist should contain 12 different titles
- ”Duration” constraint: the playlist should not last more than 76 minutes
- ”Continuity” constraint: the genre of a title should be close to the genre of the next title
- ”Progression” constraint: the sequence should contain titles with increasing tempo, etc.

Solutions to this kind of problem, using titles from our 20,000-title catalogue, can be found in section 3. We have shown that this approach has many advantages over the traditional, individual title-based approaches. First, it is much easier for users to specify properties of playlists, seen as temporal sequences, than to specify properties of individual titles. Secondly, the combination of well-designed sequence properties makes it possible to produce sequences which provide optimum compromises between *desire for repetition* and *desire for surprise*, which are identified as central ingredients of ”interesting” music recommendations. Lastly, automatic playlist generation is a powerful way of exploiting systematically large catalogues, as demonstrated in [1].

However, automatic playlist generation comes with a price. We have shown that the problem of generating a playlist given a catalogue of titles with musical metadata, and a set of arbitrary constraints is a combinatorial problem. It its full generality, the problem is NP-hard, as it boils down to a finite domain constraint satisfaction problem. We proposed initially in [1] a proof-of-concept based on a complete search algorithm using finite domain constraint satisfaction techniques, and a set of specially designed filtering procedures. This approach demonstrated the power of constraints for generating playlist, but suffered from two drawbacks. First the technique did not scale up to large catalogues, especially when dealing with combinations of conflicting cardinality and distribution constraints. Second, the definition of filtering procedures for global constraints required the design and implementation of tricky look-ahead techniques, limiting in practice the expressiveness of the system.

Alghoniemy & Tewfik ([2, 3, 4]) have proposed an alternative approach to the playlist generation problem, based on integer linear programming and branch & bound techniques. Their approach is able to handle only two types of constraints, and is limited to Boolean metadata. The authors report results on catalogues of limited size (400), and no indication is given on the scalability, and on the possibility to introduce more complex - and realistic - constraints. Additionally, the reported performance is poor (35 seconds for generating playlists from a 400 title catalogue on a Ultra 5 platform).

Other approaches have been proposed to generate playlists based on similarity relations between music titles : the user selects a few ”seed songs”, and the system generates a playlist of songs that are similar to the seed songs. Several music similarity measures have been studied for this purpose : timbre similarity ([5, 6]), collaborative filtering ([7, 8]), or machine learning from existing playlists ([9]). These methods are intrinsically unable to cope with sequence constraints such as the ones introduced above. Furthermore, similarity relations - of any kind - can be seen as specific forms of continuity constraints. These approaches are therefore totally subsumed by the approach presented here.

In this paper, we present a novel solution to the automatic playlist generation problem which scales up and handles arbitrarily complex constraints. We report on the use of the technique on a real music catalogue with real metadata.

2. PROBLEM STATEMENT

The problem we want to solve may be defined as follows. We consider a playlist as a sequence $\mathcal{S}$ of variables $v_1, v_2 \dots v_n$ whose values x_i can be taken from a finite, possibly very large, catalogue of music titles. Each variable v_i represents the i^{th} item in the playlist. The problem is to assign values to each variable so that the resulting sequence satisfies a set of constraints chosen by the user. The constraints may hold on properties of specific attributes of the music titles. These attributes are typically musical metadata, such as genre, tempo, duration, artist, etc. The issue of extracting these metadata is out of the scope of this paper, and is addressed for instance in [10].

2.1. Adaptive Search

The technique we propose is based on an adaptation of local search techniques to constraint satisfaction, called adaptive search (see Codognet [11]). The idea is to explore the search space by refining progressively solutions, rather than by a complete enumeration. In this context, constraints are seen as simple cost functions. The cost of the constraints represents how "bad" the constraint is satisfied, for a given assignment of variables.

More precisely, we define:

- the cost $Cost(v_i, \mathcal{C})$ of a given variable v_i with value x_i, with respect to a given constraint $\mathcal{C}$, which represents "how badly" x_i satisfies $\mathcal{C}$,
- the cost $Cost(v_i)$ of a given variable v_i with value x_i, which is the weighted sum of its costs $Cost(v_i, \mathcal{C})$ with respect to each constraint holding on v_i.

The algorithm works as follows:

- Start by a random assignment of values to variables (i.e. a random sequence of music titles),
- Compute $Cost(\mathcal{S})$, the total cost of the sequence. $Cost(\mathcal{S})$ is the sum of the variables' costs $Cost(v_i)$.
- Repeat until $Cost(\mathcal{S})$ is below a given threshold:
  - For each variable v_i, compute $Cost(v_i)$, i.e. the sum of the $Cost(v_i, \mathcal{C})$ for all $\mathcal{C}$ holding on v_i
  - Find v_w, the "worst variable" in the sequence, i.e. whose cost is the highest,
  - Find its best possible new value by trying successively all the values in the catalogue, and select the value that minimizes $Cost(\mathcal{S})$,
  - Assign this value to v_w.

Additionally, there is a provision for handling local minima, through the use of a Tabu list: worst variables for which no value can be found to decrease the total cost are labelled as Tabu for a given number of iterations. This trick, along with a technique inspired by simulated annealing, forces the algorithm to explore other regions of the search space.

2.2. Definition of Global Constraints

The main interest of this algorithm is that constraints are simply seen as cost functions, and hence are very easy to define. For instance, the "all different" constraint stating that all variables should have different values is defined as follows:

```
AllDifferentCt.cost()
Return 1 - the number of different values in the
sequence divided by the size of the sequence
```

More complex constraints can be defined as easily. For instance, the cardinality constraint on genre (e.g. "the playlist should contain at least 60% of instrumental titles") is defined as follows:

```
CardinalityCt.cost()
Return abs(actual number of values that have
the wished attribute - wished number of values)
divided by the size of the sequence
```

In practice, the cost functions are defined more efficiently, by passing as argument the lastly modified variable to the cost functions. This information is used to compute only the differential cost, instead of the whole cost.

2.3. Pre-filtering

With certain constraints, e.g. "equality" ("item #2 should be a song by the Beatles"), or "set membership" ("the genres of items #5 to #8 should be one of {jazz, rock}"), the previous algorithm clearly does not need to search through all values in the catalogue. Therefore, we add a pre-filtering phase in which each variable v_i is linked to a search domain d_i which is the whole catalogue by default, but can also be a subset of the catalogue. For the previous examples, d_2 is the set of all songs by the Beatles, and $d_{5\dots8}$ is the subset of all titles of jazz and rock. These domains can be obtained very efficiently with existing database search techniques, e.g. a simple "select" query in a SQL database. Then, the later adaptive search algorithm only has to look for possible values of a variable v_i in its domain d_i.

3. EXPERIMENTS

3.1. The database

Experiments were performed in the context of the Cuidado European IST project ([10]). In this project we have setup a database of 17,075 popular music titles, together with metadata extracted automatically through different techniques. Metadata for each title includes duration (in seconds), subjective energy (a float between 0 and 1), tempo (an integer between 60 and 180 bpm), artist, genre, instrumentation (e.g. "guitar", piano", "singing"), performer configuration (e.g. "Rock band", "a capella", "Man", "Woman", etc.), vocal quality, etc.

3.2. Example

We give here a typical example of playlist generated by our system : a 10 title playlist with "All Different", increasing tempo, two cardinality constraints on genre (50% Folk,50% Rock), genre continuity from item to item ("Folk/Rock" - "Rock" is better than "Folk/Blues" - "Rock"), and genre distribution (items of the same genre must be as separated as possible from one another).

```
1) Free Ride - Nick Drake - tempo 64 - genre Folk
\ Pop
2) Shadowboxer - Fiona Apple - tempo 71 - genre
Rock \ Folk \ Alternatif
3) The Future - Leonard Cohen - tempo 72 - genre
Folk \ Pop
```

```
4) To your love - Fiona Apple - tempo 80 - genre
Rock \ Folk \ Alternatif
5) City of New Orleans - Arlo Guthrie - tempo 85
- genre Folk \ Rock
6) Carrion - Fiona Apple - tempo 88 - genre Rock
\ Folk \ Alternatif
7) Tom's Dinner - Suzanne Vega - tempo 123 -
genre Folk \ Pop
8) Talkin about a revolution - Tracy Chapman -
tempo 125 - genre Rock \ Folk
9) The Boy In The Bubble - Paul Simon - tempo 138
- genre Folk \ Pop
10) Quand plus rien ne va - Mes souliers sont
rouges - tempo 144 - genre Rock \ Alternatif \
Folk
```

An exact solution was found in 256 iterations, and took 4 seconds. It is easy to check that the genre cardinality is correct (5 folk, 5 rock), that the tempo is increasing and the genre distribution constraint is also well satisfied. The two constraints of genre continuity and distribution are contradictory, however the system has found a solution by selecting the right subgenres (Folk/Rock and Rock/Folk).

3.3. Quantitative evaluations

We report here systematic results on the convergence time of the search algorithm, using the same constraint set as above in 3.2.

- For a given set of constraints, increasing the size of the playlist (from 4 to 50) does not increase significantly the convergence time. The search algorithm does not explore the sequence space extensively, but starts at random and execute a number of "best local moves", which number does not seem to grow with the sequence's size.
- The convergence time grows linearly with the size of the database, since all possible values are tested on each iteration. Figure 1 shows the time to converge to an exact solution to the problem stated above with database sizes growing from 200 to 200,000. Solutions for sizes smaller than 1000 are computed in less than 100 ms, which is several hundred times faster than previous attempts [1, 2, 3, 4]. For our 20,000 item database, the mean convergence time is 1.45 second. To show that the technique scales up to very large databases, we artificially duplicated the original database to reach up to 200,000 items, taking about 20 seconds for an exact solution. As shown in the next paragraph, this time can be greatly reduced by considering only approximate solutions.

One strong advantage of our algorithm over the previous attempts described in section 1 is that it tries to minimize a cost-function. Thus, it can give approximate solutions at any time. Figure 2 shows a typical convergence profile: the convergence speed is very fast at the beginning, and a nearly exact solution is found in very few iterations (e.g. going down from cost = 500 to cost = 2 in the first 5 iterations). Most of the total time is then spent in simulated annealing to try to find the exact solution. (found at iteration # 68 in Figure 2).

A statistical study of the mean ratio between the time to find a cost-2 approximate solution and the time to find an cost-0 exact

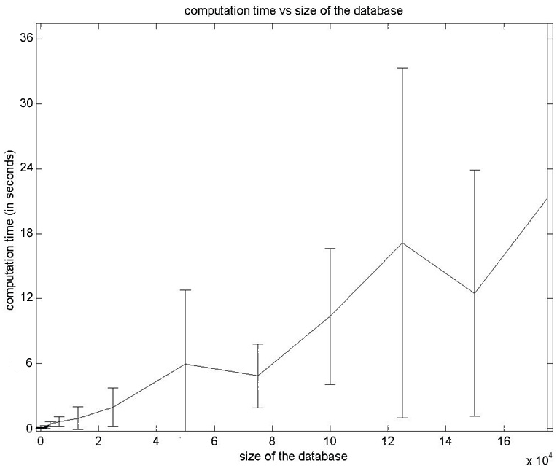

Figure 1: Convergence time vs the size of the database

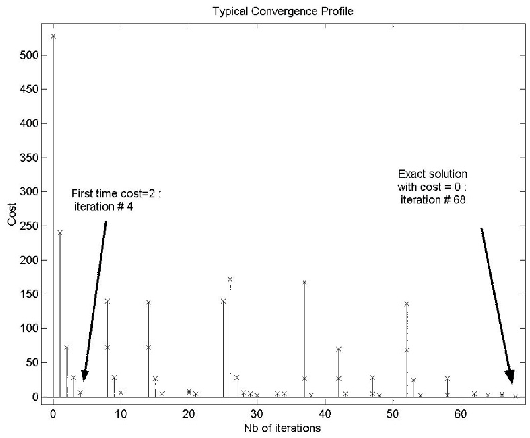

Figure 2: A typical profile of convergence, showing the costs of the best playlist vs the number of iterations so far

solution can be seen in Figure 3. It shows that regardless of the size of the database, we can improve the performance shown earlier by more than 75% by returning the first "best" approximate solution.

The following playlist shows such an approximate solution to the problem stated above, in a 150,000 title database. It can be compared with the exact solution in 3.2. The playlist is found in 3 seconds, whereas the normal convergence time as shown in Figure 1 is about 12 seconds.

```
1) Free Ride - Nick Drake - tempo 64 - genre
Folk\Pop
2) Shadowboxer - Fiona Apple - tempo 71 - genre
Rock\Folk\Alternatif
3) The Future - Leonard Cohen - tempo 72 - genre
Folk\Pop
4) To your love - Fiona Apple - tempo 80 - genre
Rock\Folk\Alternatif
5) Carrion - Fiona Apple - tempo 88 - genre
Rock\Folk\Alternatif
```

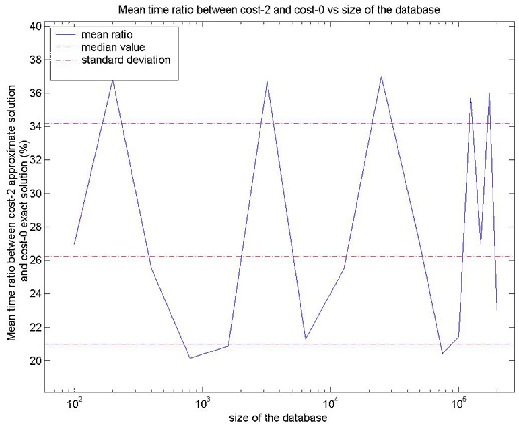

Figure 3: Ratio between the convergence times to the first "best" approximate solution and to the exact solution, plotted versus the database size. The median value is 26%

```
6) Last Night of the World - Bruce Cockburn -
tempo 119 - Genre Folk
7) Heal the world - Michael Jackson - tempo 128 -
genre Rock\Funk
8) City of New Orleans - Arlo Guthrie - tempo 85
- genre Folk\Rock
9) The Sound of crying - Prefab Sprout - tempo
118 - genre Rock
10) It's all in the game - Art Garfunkel - tempo
150 - genre Folk\Pop
```

One can see that the approximate solution has fallen in some local minima: the tempo does not increase globally, but increases on two intervals (from item 1 to 7, and again from 8 to 10). Genre distribution fails between items 3 and 4, and genre continuity fails between items 6 and 7. However, in many applications, such approximate solutions are sufficient. If more computing time is available, the system can continue to run in a separate thread, and progressively refine the solution.

3.4. A word about qualitative evaluation

The works on playlist generation based on music similarity [5, 6, 7, 8, 9] mentioned in section 1 often try to evaluate the user satisfaction with the playlists they generate (e.g. how many songs in the playlists are judged similar to the seed songs by the average user). This is in fact a qualitative evaluation of a similarity measure, not of the playlist generation process (which in itself is trivial). Since our system works with any, arbitrarily complex set of constraints (which subsumes any kind of similarity measure), we cannot evaluate any intrinsic user satisfaction. More precisely, the quality of the playlists generated with regards to some user preferences (i.e. a given set of constraints) depends on the quality of the musical metadata (descriptors and similarity measures) involved in these preferences.

3.5. Further Work

Current work focuses on the design of simple interfaces that allows the specification of complex constraints graphically. Moreover, the good performance obtained (especially the very fast convergence to an approximate solution) makes possible interactive playlist generation, through relevance feedback techniques. In this context, the user may incrementally build playlists by refining the constraint set, add or remove some songs at each step.

4. CONCLUSION

We have introduced a technique for generating playlists from large music catalogues that satisfy arbitrarily complex constraints. Our approach is based on the use of adaptive search techniques, and the design of generic constraint classes such as cardinality and distribution. We have illustrated our system with a non-optimized Java prototype on a realistic music catalogue, with automatically extracted metadata. This playlist generation facility is a key feature of the content-based music browser developed in the context of the Cuidado European Project [10]. The optimized version of our system should allow almost real time solutions on realistic music catalogues.

5. REFERENCES

[1] F. Pachet, P. Roy, and D. Cazaly, "A combinatorial approach to content-based music selection," in *Proc. of IEEE International Conference on Multimedia Computing and Systems, Firenze (It), Vol. 1 pp. 457-462*, 1999.

[2] M. Alghoniemy and A.H. Tewfik, "Personalized music distribution," in *Proc. IEEE International Conference on Acoustic, Speech and Signal Processing, ICASSP'00, Turkey*, June 2000.

[3] M. Alghoniemy and A.H. Tewfik, "User-defined music sequence retrieval," in *Proc. the eighth ACM International Multimedia Conference, Part II, Los Angeles*, November 2000.

[4] M. Alghoniemy and A.H. Tewfik, "A network flow model for playlist generation," in *Proc. IEEE International Conference Multimedia and Expo 2001 Japan*, August 2001.

[5] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *Proceedings of of IEEE International Conference on Multimedia and Expo (ICME)*, 2001.

[6] J.-J. Aucouturier and Pachet F., "Finding songs that sound the same," in *submitted to ACM Multimedia*, 2002.

[7] J. C. French and D. B. Hauver, "Flycasting: On the fly broadcasting," in *Proc. WedelMusic Conference, Firenze, Italy*, November 2001.

[8] F. Pestoni, J. Wolf, A. Habib, and A. Mueller, "Karc: Radio research," in *Proc. WedelMusic Conference, Firenze, Italy*, November 2001.

[9] J. Platt, C. Burges, S. Swenson, C. Weare, and A. Zheng, "Learning a gaussian process prior for automatically generating music playlists," *Advances in Neural Information Processing*, to appear.

[10] F. Pachet, "Metadata for music and sounds: The cuidado project," in *Proceedings of the CBMI Workshop, University of Brescia*, September 2001.

[11] P. Codognet and D. Diaz, "Yet another local search method for constraint solving," in *Proceedings of the AAAI Fall 2001 Symposium, Cape Cod, MA*, November 2001.