Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-01), Limerick, Ireland, December 6-8, 2001

MUSICAL MOSAICING

Aymeric Zils, François Pachet

Sony Computer Science Laboratory, Paris
{zils,pachet}@csl.sony.fr

ABSTRACT

This work addresses the issue of retrieving efficiently sound samples in large databases, in the context of digital music composition. We propose a sequence generation mechanism called *musical mosaicing*, which enables to generate automatically sequences of sound samples by specifying only high-level properties of the sequence to generate. The properties of the sequence specified by the user are translated automatically into constraints holding on descriptors of the samples. The system we propose is able to scale up on databases containing more than 100.000 samples, using a local search method based on constraint solving. In this paper, we describe the method for retrieving and sequencing audio samples, and illustrate it with rhythmic and melodic musical sequences.

1. INTRODUCTION

There is a natural tendency among modern musicians to compose music by assembling pieces of existing material (sounds samples) rather than entirely from scratch. This tendency is particularly strong in the community of Techno or Dance music, but is also becoming commonplace in other musical genres, in which musicians make extensive use of computers, through the use of software such as sequencers (e.g. Cubase) or sound production systems (e.g. ProTools).

Composing music by assembling samples manually raises difficult problems when database get very large, which are not adressed by traditional software. First, it is difficult to find individual samples in large libraries, because samples are difficult to label and classify. Typically, the user manually selects the sounds he/she wants to use or reuse, and locates them in the music sequence he/she is building. Moreover, the size of sample databases is such that it is hardly possible to know all the sounds it contains and therefore to retrieve by hand the correct samples.

Secondly, samples are usually not treated in isolation, and must satisfy various properties and constraints between each other, to fit into the sequence. For instance, a composer may want to ensure some timbral continuity or discontinuity between successive samples, or may want to produce a given melody, etc. These constraints have to be enforced manually by the composer.

We propose a mechanism which allows to handle both problems (sample retrieval and sequence generation) at the same time in a computer-assisted way.

The general problem of classifying and retrieving audio samples is the subject matter of the Mpeg7 standardization effort. This problem relies on the extraction of relevant low-level features out of music samples, which has been addressed by many researchers, such as Wold [1]. The general idea of combining retrieval and sequence generation was introduced by Pachet [2], in the context of music titles sequencing. In addition, the retrieval system used complete search methods which do not scale-up to very large databases. The notion of constraints on sounds was addressed by Schwarz [3], who proposed a system which handles continuity constraints on samples. Our music composition system offers more constraints than continuity. As we will see, many other types of constraints on sounds may be useful when composing, such as difference, cardinality, distribution, etc.

The mechanism we propose is called *musaicing* (for *mu*sical mo*saicing*), by analogy with image mosaicing, which consists in building an image by assembling a large number of small images (see Robert Silvers' Photomosaics [4]). This approach exploits our capacity to synthesize a perception at a macro level of an image or a melody, from the cumulated perception of items at the micro levels such as pixels or samples. Similarly to image mosaicing, musaicing allows to build a musical sequence by specifying global properties of the sequence, and letting the system select and sequence automatically the sound samples.

Musaicing can be used to compose sequences with arbitrary high-level properties, as we will see below, but can also be used to produce imitations of target sound sequences, as in image mosaicing. For example, a musical mosaic can be made from a song of the Beatles, recomposed from small extracts of the most famous rock titles of the 60ies. This approach provides two levels of listening: a distant listening which "sounds like" the original Beatles song, and a closer listening which reveals the different extracts of the 60's titles that make up the mosaic.

This principle of composing music by sequencing samples has been exploited e.g. by composer John Oswald [5], who manually copied and pastes the samples. The musaicing system we propose here is a generalization of this approach to handle arbitrary sample characteristics and sequence properties.

2. THE MUSAICING MECHANISM

In this section, we present how the system produces sound sequences, which we call *musaics*, out of large databases of samples. As explained above, musaics are made up of individual segments, where each segment is a sound sample from a given database. The generation problem is seen as a constraint problem on the properties of the whole sequence, as well as the properties of each segment that constitutes it. By selecting the right samples, we assign values to the segments so that all the constraints specified by the user are optimally satisfied. In a nutshell, the generation problem is defined by a set of *variables* (the segments

of the musaic), a set of *constraints* weighted along their importance, with their associated *cost functions* which aggregate the constraints costs on variables, and a *global cost function* to minimize.

2.1. Definition of the sequence properties

Any wished property of the sequence can be translated into a specific constraint. There are two types of constraints: *segment constraints* and *sequence constraints.*

A segment constraint is a local constraint on one specific segment of the sequence, related to one specific descriptor. The segments are described using a set of descriptors, such as:
- their mean pitch (calculated using Zero Crossing Rate),
- their loudness (calculated using energy),
- their percussivity (calculated using amplitude variations),
- their global timbre (calculated using spectral distribution),...

A segment constraint imposes a target value for one descriptor, for example a given pitch or a given percussivity.

A sequence constraint controls a property of the global sequence. These constraints can apply any set of segments, which can be the whole sequence. We have defined several such constraints:
- distribution constraints are the most important constraints for sequencing problems: they control the location of different samples in the sequence, according to their properties. For instance, we can use them to impose regular rhythmic patters, by specifying a distribution of percussive sounds along the sequence,
- parameters continuity constraints control the continuity of segments according to a specific dimension or feature. For example, the pitch continuity constraint reduces the difference between pitches of consecutive samples, and the title continuity constraint aims at choosing consecutive samples in the sequence that are also consecutive in the original song from which they are extracted. The system proposed by Schwarz [3] is based on parameters continuity between successive elements.
- cardinality constraints deal with the number of different samples in the sequence, allowing to control the uniformity or the variety of the samples in the musaic according to specific parameters. The most basic cardinality constraint is the 'All-Different' constraint, but they can be as diverse as '80% of percussive sounds', or 'pitches follow a distribution centered around 440 Hz',...

Contrarily to segment constraints, which are limited by the number of descriptors of the segments, the sequence constraints can be of any type, and any user can easily build up new ones.

Once all the constraints are defined, the system provides another control on the sequence by associating weights to them. These weights represents how important it is to satisfy the constraint during the sequencing. For example, on a 0-100 scale, setting the 'Pitch constraint' weight to 100 and the 'All-different' weight to 50 means that it is twice as important to obtain correct pitches than to have only unique samples. These weigths define the priority when all the constraints can not be solved simultaneously, which is the most frequent case.

There are two ways to generate the constraints. In manual mode, the user selects all the constraints by hand, whereas in imitation mode, the segment constraints are automatically specified from a target song the user wants to rebuild: the musaic will be made of samples with the same properties as the original title's, i.e. the same descriptors' values. These local segment constraints are sufficient to build a musaic, but there is no guarantee on the global structure of the sequence. Therefore, to cope with this problem, the user can specify additional sequence constraints, such as continuity or distributions, etc…

2.2. Satisfaction of the sequence properties

The system aims at generating a musaic that satisfies all the constraints defining the sequence properties. But most of the time, the constraints cannot all be satisfied at the same time: we need to introduce a measure of the distance between the musaic and the desired sequence. Each constraint is therefore associated to cost functions evaluating its satisfaction for each segment. Building the musaic then consists in minimizing all the constraints cost functions at the same time, to fit all of the properties simultaneously.

2.2.1. Segment constraint cost function

In the case of a segment constraint, the cost function is the difference between the segment descriptor's value and the specified target value, normalized between 0 and 1. For instance, the cost of a 'pitch' constraint, imposed on the segment 'segment' that contains the sample 'sample', is defined as follows:

$$PITCH\_COST(segment) = \frac{\left|PITCH(sample) - TARGET\_PITCH(segment)\right|}{\underset{s \in DOMAIN(segment)}{MAX}\left(\left|PITCH(s) - TARGET\_PITCH(segment)\right|\right)}$$

The normalization is done by dividing the pitch difference by the maximum pitch difference between two samples.

2.2.2. Sequence constraint cost function

In the case of a sequence constraint applied on a set of segments, we need to build up a cost function that evaluates the contribution of each segment to the global satisfaction of the constraint, and returns a normalized result between 0 and 1. For instance, the cost of the 'All different' constraint, applied on the whole sequence 'sequence', is defined as follows for the segment 'segment' containing the sample 'sample':

$$ALLDIFF\_COST(segment) = \sum_{s \in sequence, s \neq sample} isequal(sample, s)/(length(sequence) - 1)$$

If all the samples in the sequence are the same as 'sample', the 'all diff' cost is 1, the constraint is not satisfied at all. On the contrary, when no other segment contains 'segment_sample', the constraint is satisfied, and the cost is 0.

For a distribution constraint, the cost takes the location of the sample into account. Here is an example of 'percussive tempo' constraint, which imposes regular percussive sounds along the sequence. This constraint has 2 control parameters : the tempo, between 50 and 180 bpm, which controls the period between 2 consecutive percussive sounds, and the phaseshift, between 0 and 2π, which controls the position of the first percussive sound in the sequence. The constraint cost is defined as follows, for the segment 'segment' containing the sample 'sample':

* the system first computes the location of the segments that have to be percussive, using the tempo and the phaseshift, and sets their tempo index to 1; the tempo indexes of the others segments are set to 0.
* the cost is then computed as :

$$PERCUSSIVE\ TEMPO\_COST(segment) = \begin{cases} \dfrac{\underset{samples}{MAX}(Percussivity) - PERCUSSIVITY(sample)}{\underset{samples}{MAX}(Percussivity)}, if\ (index(segment) = 1) \\ 0, if\ (index(segment) = 0) \end{cases}$$

The cost diminishes as the percussivity of the indexed segments increases, and reaches 0 when all these indexed segments have the highest possible percussivity. Note that the constraint does not take the non-indexed segments into account, therefore they can take any value without affecting the cost.

2.2.3. From constraint costs to segment cost

The constraint cost functions evaluate the satisfaction of one specific constraint for each segment. The solving of the problem requires also the definition of *segment costs*, which combine and evaluate the satisfaction of all the constraints related to one segment. These segment costs represent the contribution of the segment to the global cost of the sequence, i.e. whether a segment fits well the sequence properties. These segment costs take each constraint into account according to the constraint weight that represents the importance of the constraint satisfaction:

$$COST(segment) = \sum_{Constraints} WEIGHT(Constraint) * CONSTRAINT\_COST(segment)$$

These segment costs enable to detect the segments that least fit the sequence, in order to replace them iteratively during the building of the musaic.

2.2.4. From segment costs to global cost

Finally, we build a global cost function, which is an aggregation of all the constraints costs, computed by summing all the segment costs of the musaic segments. That global cost function represents the distance between the current musaic and the desired sequence:

$$GLOBAL\_COST(sequence) = \sum_{Segment \in sequence} COST(segment)$$

This is the criterium which evaluates the global satisfaction of all the constraints, and which has to be minimized during the building of the musaic.

2.3. Building the musaic

Building the musaic consists in finding a sequence out of a database of samples that best satisfies all the sequence properties defined by the constraints.

When dealing with very large databases of samples, a complete search method is absolutely prohibited in order to obtain quick results. So the constraint system is solved using a local search algorithm called "Adaptive search" [6][7], which was shown to be well adapted for large-scale combinatorial problems for which optimality is not required. Indeed, in the case of music, the interesting point is that, if no exact solution exists, there are many approximate solutions which can provide very different and interesting results. For example, an application of this local search method to music composition on symbolic values (pitches) is investigated by Truchet [8], who has shown that the method outperforms complete search methods using CSP techniques. The "Adaptive search" algorithm is defined as follows:

1. *Random initialization*

REPEAT

2. *Compute all non-tabu variables costs*
3. *Select the highest-cost variable Vh*
4. *Search for the value v of Vh inducing the lowest global cost*
5. *If the global cost cannot be improved, mark Vh as 'tabu'*
6. *If all the variables are tabu, random reinitialization*

UNTIL (global cost<e) or (max iterations reached)

In the case of musaicing, we have added a preliminary step to the "Adaptive search" that consists in defining the variables domains, i.e. all the possible samples for each segment of the musaic. The user controls the size of the domain by specifying it in the sequence properties. For each segment, the domain is chosen by setting a restriction on the values of the descriptors linked to the segment constraints. The domain definition also takes the weights of the segment constraints into account: the higher the weight, the smaller the domain. Once the domains are defined, the "Adaptive search" algorithm applied to musaicing is:

1. *Random initialization :*
   The sequencing starts with an initial sequence built up from a set of samples taken randomly in the predefined domains, evaluates all the constraint costs related to the current sequence, and computes the global cost that has to be minimized,

REPEAT

2. *Compute all non-tabu variables costs:*
   i.e. the segments costs related to each sample of the sequence,
3. *Select the highest-cost variable :*
   Selection of the 'worst' segment that induces the highest constraint cost, i.e. that is the most responsible for the distance between the current musaic and the desired sequence,
4. *Search for the value inducing the lowest global cost:*
   Browse the database of samples in order to find whether another sample that better fits the constraints could replace

the 'worst' segment, and consequently diminish the constraints costs and the distance from the desired sequence,

5. *If the global cost cannot be improved, mark Vh as 'tabu':*
   If the sample can not be replaced to improve the sequencing, this significates that the current configuration of the sequence does not allow to obtain a better local result around this segment, so the 'worst' segment is marked as 'tabu'. That means that the segment can not be selected as the 'worst' segment for a few iterations, during which the sequence configuration can change,
6. *If all the variables are tabu, random reinitialization*:
   If all segments are 'tabu', the sequence has reached a stable local minimal cost configuration depending on the initial random sequence, and no changes of the samples can improve the sequencing: a new random initial samples sequence is computed. Indeed, as the process is very quick, we can repeat the "adaptive search" algorithm several times, by taking at each time a new random initial sequence,

UNTIL the global cost is lower than a predefined threshold, or the maximum number of iterations is reached. The final musaic is the best sequence of all these attempts, i.e. the sequence that has the lowest final global cost.

Once the solution is found, the musaic quality can be improved by a *sequence refining*, consisting in applying global transformations on transitions, in order to smoothen possible rough transitions between samples.

3. EXAMPLES

3.1. Basic example

First we present a basic example that clearly shows how the algorithm works and takes the constraints and their associated weights into account. The goal of this example is to build a musaic imitating a synthesized target recording characterized by a pitch that *increases* continuously from 0 to 1000Hz.
The database contains samples extracted from 50 pop and rock songs, including a set of samples extracted from 1 synthesized recording with the same characteristics as the target, except that the pitch *decreases* continuously from 1000 to 0Hz.

3.1.1. Pitch constraint

First, we impose only one pitch constraint on all the segments, imitating the increasing pitch of the target sequence. The constraint weight is set to 50. Figure 1 shows the pitch analysis of the musaic:

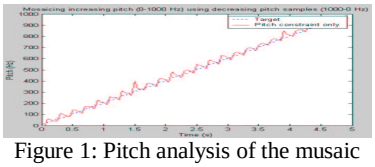

Figure 1: Pitch analysis of the musaic

First, as figure 1 shows, the pitch constraint is globally satisfied: the pitch increases continuously, and is close to the target pitch.
Secondly, the imitation is good, in the sense that the samples selected by the system are extracted from the pitch decreasing synthesized recording, which has the closest characteristics from the original target. This property is obvious on the figure 1, where the little 'pitch sawteeth' show that the selected samples have all a decreasing pitch.

3.1.2. Pitch and Continuity constraints, equal weights

We can complexify the previous example by adding a continuity constraint on all the segments. That continuity constraint goes in the way of the pitch constraint. Indeed, the pitch constraint imposes an increasing pitch, whereas the continuity constraint requires the use of longer extracts from the database, whose pitches decrease. Figure 2 shows the pitch analysis of the musaic when the weights of the two constraints are set to 50:

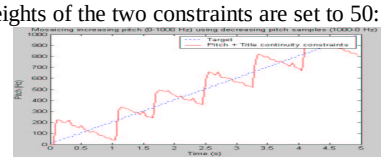

Figure 2: Pitch analysis of the musaic

The pitch constraint is globally satisfied: the pitch increases, but not continuously anymore.
The continuity constraint is locally satisfied: the system tries to use as much as possible contiguous samples from the database, and we can see larger pitch decreasing parts. Indeed, since the two constraints are conflicting, the system finds a balance between them by building 'pitch sawteeth'. The pitch globally increases, but sometimes locally decreases.

3.1.3. Pitch constraint weight < Continuity constraint weight

Finally, we can experiment with the influence of the weights of the constraints on the resulting musaic. The continuity constraint's weight is now raised to 100, whereas the pitch constraint's weight stays at 50. Figure 3 shows the pitch analysis of the musaic:

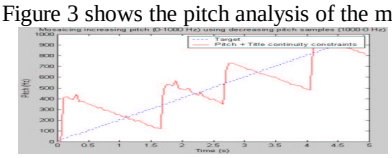

Figure 3: Pitch analysis of the musaic

The pitch constraint is very slightly satisfied: the pitch only shows a global long term increasing.
On contrary, the continuity constraint is very well satisfied: the sequence has been divided in only 4 parts, represented by the bigger 'pitch sawteeth'. Indeed, the system still finds a balance between the two conflicting constraints by building 'pitch sawteeth'. But the balance has been modified by applying a higher weight to the continuity constraint. The satisfaction of the continuity constraint is given priority over the satisfaction of the pitch constraint, resulting in larger 'decreasing pitch sawteeth'.
Hence, the user can control the size of the "teeth" with the relative weights of the constraints.

This example first shows that, in the case of conflicting constraints, the system takes both constraints into account, and finds a balance between them. The second conclusion is that the user has a complete control on that balance, by setting the relative weights of the constraints.

3.2. Percussive tempo

As explained in 2.2.2., the percussive tempo is an example of sequence constraint that combines features retrieval and samples distribution modeling: the system uses the percussivity feature to find the most percussive sounds and their location is controlled by a distribution model. That constraint consists in specifying two parameters : a tempo and a phaseshift, and intends to set percussive samples in the musaic according to them.

Figure 4 shows an example of percussive tempo 80 with null phaseshift:

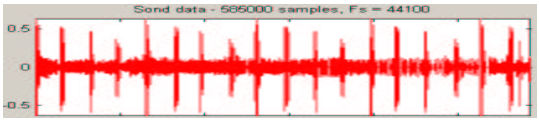

Figure 4: Signal of the musaic

The constraint is clearly satisfied: we can see regular peaks on the signal, corresponding to the percussions sounds., with an evaluated tempo of 78. The difference between the specified tempo and the musaic tempo is due to the length of the musaic samples.

We can also add a phaseshift to the constraint. Figure 5 shows an example of percussive tempo 130 with phaseshift=$\pi/2$:

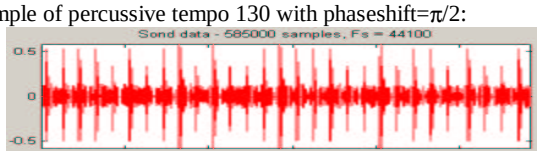

Figure 5: Signal of the musaic

The percussive tempo constraint is still clearly satisfied: we can see regular percussion peaks on the signal, with an evaluated tempo of 129. The phaseshift is also satisfied, as shown by the timeshift at the beginning of the signal.
These examples show the influence of the two parameters on the resulting musaic.

3.3. Combination of segment and sequence constraints

Finally, a more powerful use of musaicing is shown below, by combining different types of constraints.

3.3.1. Simultaneous Pitch and Percussive Tempo constraints

First, we impose simultaneously a pitch constraint and a percussive tempo constraint. Here is an example of a percussive tempo of 100 (with no phaseshift), with a constraint on the pitch 'increase and then decrease along the sequence'. Figure 6 shows the signal of the resulting musaic:

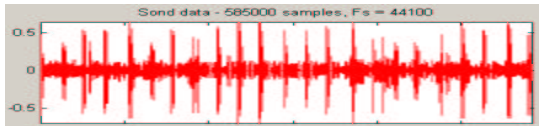

Figure 6: Signal of the musaic

Figure 6 shows that the percussive tempo constraint is well satisfied: we can see regular percussion peaks on the signal, with an evaluated tempo of 103.
To evaluate the satisfaction of the pitch constraint, we need a pitch analysis of the signal, shown on figure 7:

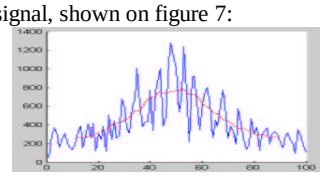

Figure 7: Pitch analysis

Figure 7 shows that the pitch globally satisfies the constraint 'Increase then Decrease'. The local irregularities are due to the percussive sounds, whose pitches are not significant. However, the fact that even the extremal pitch values 'increase then decrease' shows that the pitch constraint has also an influence on the percussive samples. On contrary to pitch and continuity presented in 3.1., the two constraints are not conflicting, and can be satisfied at the same time.
This combination of constraints is interesting for a target song imitation. Indeed, the pitch constraint is interesting for melody imitation: with an efficient pitch analysis, the system is able to build a musaic that follows the melody of the target song. The additional percussive tempo constraint gives the possibility to link a rhythmic structure to the musaic.

3.3.2. Pitch and 2 Simultaneous Percussive Tempo constraints

We can obtain even more precise control on the musaic by combining several constraints of the same type, but with different parameters. For instance, we can have a more precise percussive rhythm by controlling separately the different types of drum sounds. Indeed, the percussive sounds can be classified into bass-drum-like and snare-drum-like sounds using their zero-crossing rate (ZCR), as shown in [9]. An interesting rhythmic control consists in building two percussive tempo constraints, one applied on bass-drum sounds, whose ZCR is low (LF), and the other applied on snare-drum sounds, whose ZCR is high (HF). These two constraints are specified simultaneously with different parameters. For example, we can build a high-level rhythmic structure, with bass-drum sounds following a fast tempo (132), and snare-drum sounds following a slower tempo (2/3 of the previous tempo = 88), so that snare-drum sounds occur regularly every 1.5 bass-drum sound. In order to avoid imultaneous sounds conflicts, we introduce a slight delay (phaseshift =$\pi/2$) for the snare-drum sounds. In addition, we added the pitch constraint 'Increase then Decrease'. Figure 8 shows the resulting musaic:

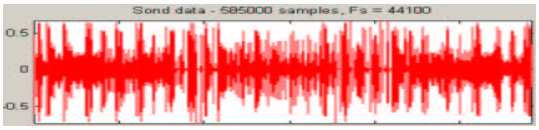

Figure 8: Signal of the musaic

The signal is now too complex to show explicit properties of the musaic, we need to use filters to observe its different components.

Figure 9 shows the low-pass filtered part of the signal:

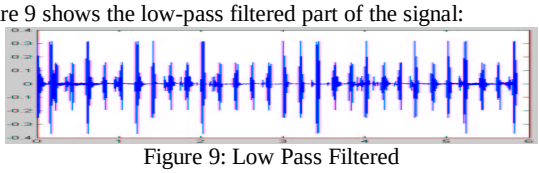
Figure 9: Low Pass Filtered

Figure 9 shows regular percussion peaks on the signal, with an evaluated tempo of 131. These low-frequency peaks correspond to the bass-drum percussion sounds, showing that the percussive tempo constraint on LF is satisfied.

Figure 10 shows the high-pass filtered part of the signal:

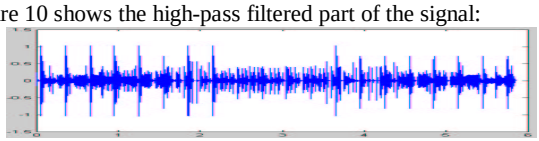
Figure 10: High Pass Filtered

Figure 10 shows regular percussion peaks (that are less visible in the middle part of the signal), with an evaluated tempo of 87. Similarly to figure 9, these high-frequency peaks correspond to the snare-drum percussions, showing that the percussive tempo constraint on HF is also satisfied.

In addition, the pitch analysis on figure 11 shows that the pitch still globally satisfies the constraint 'Increase then Decrease':

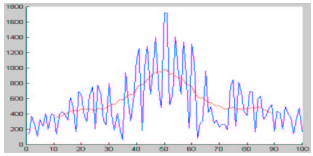
Figure 11: Pitch analysis

The combination of these 2 percussive tempo constraints provide a complex rhythmic structure, using a (B)ass / (S)nare sequence following the drum track : [BS__B__SB___]. That rhythmic structure can easily be controlled by changing the tempo and phaseshift of the constraints.

These examples show how combinations of constraints provide efficient controls on the sequencing. These controls can be very useful for musical composition, which can either be done from scratch or be based on the imitation musaic of an preexisting musical recording. The examples shown in the paper can be heard at: http://www.csl.sony.fr/Music.

4. CONCLUSION

We have introduced a method for retrieving and sequencing samples out of a large database. This method spares to the user the task of selecting and locating individual samples. Instead, the user specifies high level properties of the targeted sequence. These properties are interpreted by a constraint solver. The examples show the power and flexibility of the approach. The expressive power of musaicing is determined by the nature of the available constraints, but the algorithm presented here makes it very easy to define new constraint classes.

Current work consists in coupling the system with a more refined segmentation of source and target music titles into samples, including the detection of stable notes, percussions and singing voice. Secondly, a more robust analysis of the features of the samples is under development, including more accurate pitch tracking as well as instruments and voice characterization. These improvements open the door for new kinds of audio transformation, such as reinstrumentation, e.g. replacing guitar by organ, drums by congas, etc. Finally, thanks to the efficiency of the constraint solver, we envisage the use the system in a real time context. This will allow users to change the parameters of musaicing (adding, removing constraints, changing the various weights) in real time, while listening to the result. Such a real time musaicing would allow to further reduce the gap between composition and listening, and therefore make composition accessible to a larger audience.

5. REFERENCES

[1] Wold, E., Blum, T., Keislar, D., Wheaton, J. "Content-based classification, search, and retrieval of audio", IEEE Multimedia 3(3), 27-36, Fall 1996.

[2] Pachet, F., Roy, P., Cazaly, D. "A Combinatorial Approach to Content-based Music Selection", IEEE Int. Conf. on Multimedia Computing and Systems, Firenze(It), 1999.

[3] Schwarz, D. "A system for data-driven concatenative sound synthesis", DAFX00 Proceedings, Verona (It) Dec. 2000.

[4] Robert Silver "Photomosaics", http://www.photomosaics.com

[5] John Oswald "Plunderphonics", Fony, http://www.plunderphonics.com

[6] Codognet, P. "Adaptive search, preliminary results", proceedings ERCIM / CompulogNet Workshop on Constraints, Venise, Italy, June 2000.

[7] Codognet, P., Diaz, D. "Yet another search method for constraint solving", AAAI Symposium, North Falmouth, Massachusetts, November 2001.

[8] Truchet, C., Agon, C., Assayag, G. "CAO et contraintes", Proceedings of 8th Journées d'Informatique Musicale, Bourges (France), GMEB, 2001.

[9] Gouyon F., Pachet F., Delerue O. "On the use of zero-crossing rate for the classification of percussive sounds in polyphonic signals", DAFX00 Proceedings, Verona (It) Dec. 2000.