Ringomatic: A Real-Time Interactive Drummer Using Constraint-Satisfaction and Drum Sound Descriptors

Jean-Julien Aucouturier
SONY CSL Paris
6, rue Amyot
75005 Paris, France
jj@csl.sony.fr

François Pachet
SONY CSL Paris
6, rue Amyot
75005 Paris, France
pachet@csl.sony.fr

ABSTRACT

We describe a real-time musical agent that generates an audio drum-track by concatenating audio segments automatically extracted from pre-existing musical files. The drum-track can be controlled in real-time by specifying high-level properties (or constraints) holding on metadata automatically extracted from the audio segments. A constraint-satisfaction mechanism, based on local search, selects audio segments that best match those constraints at any time. We report on several drum track audio descriptors designed for the system. We also describe a basic mecanism for controlling the tradeoff between the agent's autonomy and reactivity, which we illustrate with experiments made in the context of a virtual duet between the system and a human pianist.

Keywords: interaction, drumtrack, metadata, constraint satisfaction, concatenative synthesis

1 INTRODUCTION

State-of-the-art sample-based drum machines (or virtual drumkits) such as Fxpansion's BFD (BFD, FXpansion Audio UK Ltd, 2003) or Toontrack's Drumkit From Hell (DFH, 2003) offer drum programmers almost total control over the sampled sounds that are played, the microphones used, the drumkit manufacturer, and even the individual drums and cymbals being used. Like other sampled instruments, they benefit from the improvement of digital storage, often offering tens of thousands of sounds from tens of different drumkits, recorded by tens of different drummers, each using several velocities for each stroke. They also ship with large libraries of Midi-like drum patterns (or presets, or "grooves"), which can be associated with one of the very many sets of sounds to give an instant, realistic drumtrack.

While the expressive power of such machines for drum programmers is unprecedented, they offer very little possibilities for interactive music systems, as proposed e.g. in Rowe (1993). On the one hand, the sounds and patterns are mostly undescribed, only using editorial, arbitrary metadata (e.g. what are the perceptual qualities of "retrobreaks fill A" ? Is it energetic ? Syncopated ? How does "kick-Leedy" sounds compared to "kick-PearlB" ?). This makes high-level mappings between a real-time musical input and the virtual drummer difficult. On the other hand, it is difficult to add new sounds into the system, in order to adapt to specific musical contexts. This typically involves buying expensive, pre-built extension packs.

In this work, we propose a sampled-based drum machine whose output can be controlled in real-time by high-level properties such as energy, density, saliency of drums or cymbals, etc. We use audio analysis techniques inspired by MIR to both gather the sampled material, which is automatically extracted from pre-existing musical files (e.g. drum solo parts in a jazz mp3) and index each sample with acoustic metadata, automatically extracted from the signal. The typical sample used in the system is a few beats' audio extract from a drum part, which correspond to a musical bar, and can therefore be looped while preserving a feeling of steady beat and metric. As seen in Figure 1, the drumtrack produced by the drummer is a continuous con-

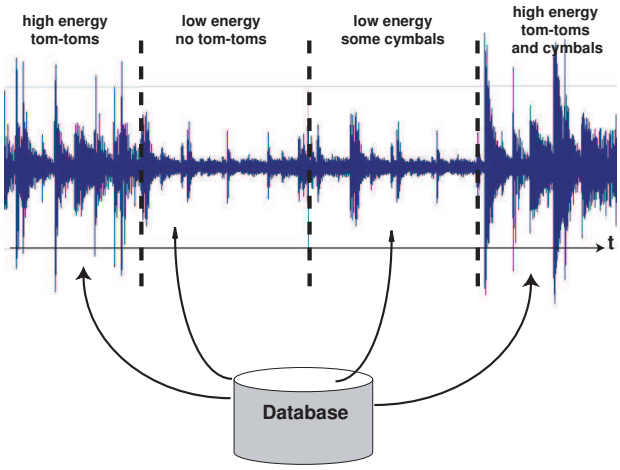

Figure 1: The drumtrack is produced by concatenating drumbars selected in a database according to their metadata.

catenation of such bars of drumming, which we call here *drumbars*. We propose a constraint-satisfaction algorithm to control the drumtrack's high-level properties, such as its energy or its continuity, and a real-time mechanism to allow constraints to be modified at any time.

Our system builds on several prior works. Rhythm is a well-covered subject of study in computer music. Bilmes (1993) works on transcriptions to study velocity and timing deviations in human performance, aiming at building more realistic drum machines. There have been numerous attempts at generating interesting rhythms, notably with Genetic Algorithms (Pachet, 2000; Tokui and Iba, 2001). On the audio side, drum sounds have received recent attention in the MIR community, to either transcribe drumtracks from polyphonic music (Zils et al., 2002), measure similarity between drum patterns (Paulus and Klapuri, 2002) or classify drum sounds, notably between bass and snare drum (Herrera et al., 2002; Yoshii et al., 2004). However, most of the work so far has focused on the timbre of individual drum strokes, rather than on perceptive qualities of full drum loops like in this paper.

Concatenative synthesis (Lazier and Cook (2003); Schwarz (2003); Zils and Pachet (2001)) is also gaining more and more attention in the field of music. Concatenative synthesis uses a database of samples, or *units*, and a *unit selection* algorithm that finds the sequence of units that match best a target sound or phrase. It mainly focuses on the precise reconstruction of the target, e.g. for realistic instrument synthesis. While our work uses the same basic scheme of concatenating sound samples selected from a database, we target an interaction context where there is no pre-specified target sequence. On this respect, it is more on the side of the automatic sampler described in Aucouturier et al. (2004).

Constraint satisfaction programming (CSP) finally is a paradigm for solving difficult combinatorial problems, particularly in the finite domain. In this paradigm, problems are represented by variables having a finite set of possible values, and constraints represent properties that the values of variables should have in solutions. CSP is a powerful paradigm because it lets the user state problems declaratively by describing a priori the properties of its solutions and use general-purpose algorithms to find them. There have been numerous applications of CSP to music, e.g. for automatic generation of playlists of music titles (Aucouturier and Pachet, 2002), automatic harmonization (Pachet and Roy, 2001) and spatialization (Pachet and Delerue, 2000). In Zils and Pachet (2001), we introduced the concept of musical mosaics ("Musaicing"), and the idea of using CSP to generate audio sequences of sound samples, with high-level constraints holding on the metadata of the samples. The work presented on this paper is a real-time, interactive extension of Musaicing.

2 AUTOMATIC GATHERING AND INDEXING OF AUDIO MATERIAL

This section describes both how drumbars are gathered automatically from existing music titles, and how each drumbar is described with automatically computed metadata. Both steps rely on EDS (Zils and Pachet, 2004), an audio classification system developed at Sony CSL.

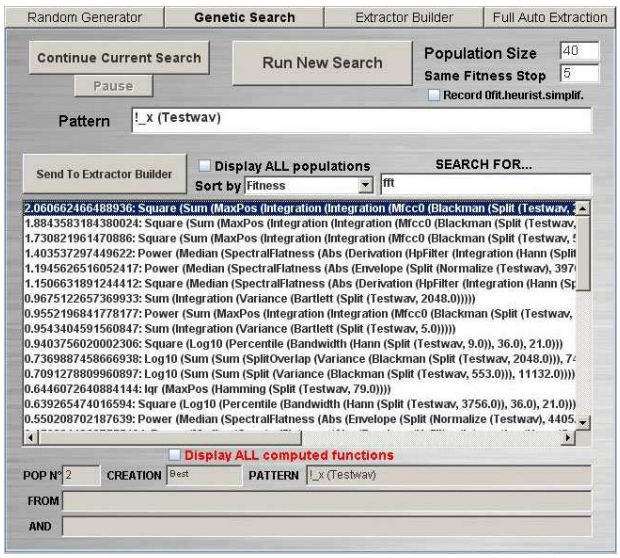

Figure 2: Screenshot of the EDS system showing the results of feature discovery for the detection of drum solo

2.1 Metadata Extraction with EDS

EDS (Extractor Discovery System) (Zils and Pachet, 2004) is a generic scheme for extracting arbitrary high-level audio descriptors from audio signals. It is able to automatically produce a fully-fledged audio extractor (an executable) from a database of labeled audio examples. It uses a supervised learning approach. Its main characteristics is that it finds automatically optimal audio features adapted to the problem at work. Descriptors are traditionally designed by combining Low-Level Descriptors (LLDs) using machine-learning algorithms (see e.g. Scheirer and Slaney (1997)). The key idea of EDS is to substitute the basic LLDs with arbitrary complex compositions of signal processing operators: EDS composes automatically operators to build features as signal processing functions that are optimal for a given descriptor extraction task. The search for specific features is based on genetic programming, a well-known technique for exploring search spaces of function compositions (Koza, 1992). Resulting features are then fed to a learning model such as a GMM or SVM to produce a fully-fledged extractor program.

2.2 Drum Solo Detection with EDS

In order to extract drumbars from music recordings, we first need the ability to detect drum solo parts, i.e. sections in music where only a drumkit is playing. This is typically a drum solo in the middle of a jazz piece or shorter drum breaks in a rock or funk song. We model the problem as a 2 class classification problem. We build a labeled database of 100 5-second music extracts, the first 50 being pure drum solo, and the other 50 various extracts of popular music, encompassing many different genres (jazz, rock, heavy metal, classical, folk, electronic), with or without drums. Figure 2 shows a screenshot of the EDS appli-

Table 1: EDS features for Drum detection

Feature	Fitness
$square(sum(maxpos(integration(mfcc0(Blackman(split(X, 2048)), 9)))))$	2.06
$power(median(SpectralFlatness(abs(envelope(split(X, 2048))))), 5)$	1.42
$mean(SpectralSpread(Split(X, 2048)))$	1.19
$sum(variance(split(X, 2048)))$	0.96
...	...
$mean(zcr(Split(X, 2048))$	0.72
$mean(SpectralFlatness(Split(X, 2048))$	0.71
$mean(SpectralCentroid(Split(X, 2048))$	0.68

cation after a few generations of 50 features and Table 1 shows the top 4 features found by EDS as well as the results achieved with some common mp7 features (zero-crossing rate, spectral flatness and SpectralCentroid), for comparison. The fitness of the features is computed with the Fisher criteria. Here are some details about the operators selected by EDS in Table 1:

- $split(X, n)$ does a n-point windowing of the signal.
- $integration$ computes the cumulated sum of a vector, i.e. $y[i] = \sum_{j=0}^{i} x[j]$.
- $mfcc0(X, n)$ computes the n first mel-cepstrum coefficients of X, including the 0th order coefficient.
- $Blackman$ is a standard Blackman window.
- $maxpos$ returns the index of the maximum value in a vector.
- $envelope$ is an envelope extractor based on the Hilbert transform.
- $Spectral\ Flatness$, $Spectral\ Spread$, $Spectral\ Flatness$, $Spectral\ Centroid$ and zcr are the standard MP7 operators.

See Zils and Pachet (2004) for more details.

The 4 best features are then fed to a number of machine learning algorithms, which are individually optimized over their parameter space (e.g. number of nearest neighbors for a knn classifier). We measure the precision by using 10-fold cross validation on the training database. As shown in Table 2, the classification results are near perfect (at best only one mis-classified instance). The best model is a k-nn classifier using 2 inverse-distance weighted nearest neighbors.

We apply the drum detector on sliding 3-second windows on full songs to segment drum solo parts. For robustness, we only look for segments corresponding to at least 3 successive windows classified as drums.

2.3 Segmentation

Once large sections of music which only includes drum solo are identified, we segment them in 4-beat drumbars using a stripped-down version of Scheirer (1998). Since beat tracking on drumtracks is usually a lot easier than on arbitrarily complex polyphonic audio, we only consider one frequency band [0-400 Hz]. A first pass is done to compute the bpm on 3-second buffers, and a second pass is done with a beat-tracker tuned on the most represented tempo found during the first pass in order to localize the beats. Then, a 4-beat-long drumbar is extracted every beat. Such a method has a number of disadvantages with respect to the metric of the musical piece. First, this makes the assumption that the signature of the piece is 4/4, which is true for a vast majority of popular music pieces, but may not always be the case. Second, the 1-beat overlap of the drumbars extraction breaks the original position of strong/weak beats. Further analysis is possible to segment more meaningful drum units, as shown e.g. in Zaanen et al. (2003).

Table 2: Precision of learning algorithms for Drum detection

Algorithm	Precision
k-Nearest Neighbor	0.99
Support Vector Machine	0.98
Neural Network	0.98
J48 pruned decision tree	0.97
Gaussian Mixture Model	0.93

2.4 Metadata Extraction

Once a database of drumbars has been gathered, we index each sample with perceptually-meaningful metadata. Again, we use the EDS system to find good specific signal processing features, and to optimize machine learning algorithms that use these features. We describe here 4 descriptors relevant for drumbars that we analyzed with EDS. For each, we give the best feature found by EDS, and the classification results using 10-fold cross validation. Each descriptor was trained on the same hand-labeled database of 75 drumbars extracted from a swing drumtrack generated by BFD (BFD, FXpansion Audio UK Ltd, 2003). In order to minimize the labeling effort, we model each problem as a 3-class classification problem (low/medium/high). However, these descriptors are intrinsically continuous, numerical values. Hence, we force the training to use a 2-nn model for classification[1], and re-use the same model as a regression model[2] in order to compute the values on the final database.

[1] i.e. a new instance is assigned to the most represented class among its neighbors

[2] i.e. the value of a new instance is the weighted mean of its neighbors values

- Energy : the perceptive energy of the drumbar, independent of the RMS volume (all drumbars are RMS-normalized). The best feature found by EDS

$$Mean(Log(Var(Split(Deriv(Square(X)), 1s)))) \quad (1)$$

is consistent with previous studies on a popular music database (Zils and Pachet, 2003). This yields a precision of 0.89.

- Onset Density : the sensation of stroke density in the drumbar. Drum rolls typically include very many strokes, while some fills may include just a few kicks and crashes. The best feature found by EDS

$$length(peaks(rms(hamming(split(X, 4096))))) \quad (2)$$

can be interpreted as a rough count of peaks of energy. The precision of the associated knn classifier is 0.92

- Presence of drums : the importance of tom and bass drum strokes as opposed to cymbals and snare drums. Jazz drummers typically use toms to give a ethnic groove to a song, rather than cymbals and ride which are typically used for swing. The best feature found by EDS

$$SpectralDecrease(Deriv(Square(Norm(X)))) \quad (3)$$

gives a classification precision of 0.84

- Presence of cymbals : the importance of high-frequency sounds like cymbals and ride. The best feature found with EDS

$$division(rms(lpfilter(X, 500, 44100)), rms(X)) \quad (4)$$

is simply the ratio of high frequency energy over the total energy of the signal. This achieves 0.82 precision.

3 Constraint-Based Concatenative Synthesis

3.1 Incremental Real-Time Constraint Satisfaction

We define the interactive generation of drumtracks as a real-time constraint-satisfaction problem (CSP). At any time, the next drumbar to be selected as well as the M latest past drumbars (M arbitrary, can be as large as $n-1$) constitute a sequence of *variables* V_{n-M}, V_{n-M+1}, ..., V_{n-1}, V_n, whose *values* can be taken from a finite database of N drumbars, called their *domain*. Each variable V_i represents the i^{th} drumbar in the sequence. We call the variable V_n corresponding to the next drumbar to be selected the *current variable*, and the V_{n-i} for $i = 1..M$ the *past variables*.

The problem is to successively assign values to each variable so that the resulting sequence satisfies a set of constraints defined by the user. The constraints may change at any time, in an asynchronous manner. Obviously, at any time, the problem can only set the value of the current variable : once a variable is played, its value cannot be changed ("one can't modify the past"). However, the choice of the next drumbar is influenced by the past choices, as constraints holding on the current variable may also hold on the past variables. The constraints typically hold on metadata of the assigned drumbars, such as the one described in section 2.4.

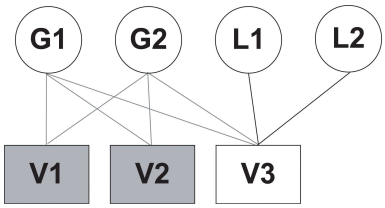

Figure 3: An incremental CSP with 3 variables and 4 constraints. V_3 is the current variable, and V_1, V_2 are the past variables.

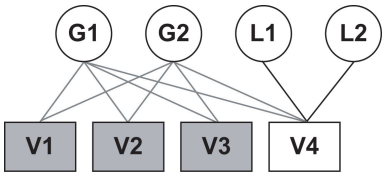

Figure 4: The same incremental CSP than in Figure 3, after the increment operation.

To model the passing of time, we introduce the notion of *increment* operation. Each time a value is assigned to the current variable, the problem is incremented, i.e. a new variable is added to the problem. The former current variable becomes a past variable, and the new variable represent the next current variable.

Figures 3 and 4 explicit the structure of the problem, and illustrate the increment operation. In Figure 3, the CSP at a given iteration i includes $M = 3$ variables, one current and 2 past, with some constraints (the G and L circles) holding on them. A value is selected for V_3, and the corresponding audio is scheduled to be played. At the next iteration $i+1$, the CSP is incremented, i.e. a new variable V_4 is added, which becomes the new current variable. Note that the scope of the constraints is automatically modified to also hold on the newly added variable. We will explicit two strategies for such a mechanism in Subsection 3.3.2 below.

3.2 Incremental Adaptive Search

The technique we propose is based on an adaptation of local search techniques to constraint satisfaction, called adaptive search (Codognet and Diaz, 2001). In our context, since only one variable can be modified at a time (the current variable), there is no combinatorial explosion of the search space. A complete enumeration of all possible values for the current variable is only the size of the drumbar database. Adaptive search is mainly targeted at off-line problems where all values must be assigned simultaneously, and a complete N^M enumeration is intractable, e.g. in playlist generation (Aucouturier and Pachet, 2002). However, adaptive search's formulation of constraints as

simple cost functions is still well suited for our problem which is clearly over-constrained : it is likely that the constraints cannot all be satisfied at the same time. The cost of a constraint represents "how bad" the constraint is satisfied, for a given assignment of variables.

More precisely, we define:

- the cost $F(V_i, C)$ of a given variable V_i with value X_i, with respect to a given constraint C, which represents "how badly" X_i satisfies C
- the cost $F(V_i)$ of a given variable V_i with value X_i, which is the weighted sum of its costs $F(V_i, C)$ with respect to each constraint holding on V_i. Each constraint has a weight, which enables to balance the importance of some constraints over some others. Section 4.2 illustrates the importance of constraint weighting.
- the global problem cost $F(CSP)$, which is the sum of the $F(V_i)$ for all V_i in the problem.

Assigning a new value to the current variable modifies the costs $F(v_n, C_i)$ of all the constraints C_i holding on v_n, and in turn modifies all the costs $F(V_i)$ of all the variables within the scope of one of several constraints C_i, and finally the global problem cost $F(CSP)$.
The algorithm works as follows:

- Start with a $n = 1$ problem, i.e. one current variable, and no past.
- Repeat :
  - Find the best possible value for V_n by trying successively all the values in the domain, and select the value that minimizes $F(CSP)$.
  - Assign this value to V_n.
  - Increment the CSP. (notably $n = n + 1$);

The first step of the repeat loop above may take a lot of time, depending on the size of the current variable's domain. However, it can be interrupted at any time, to return the best solution *so far*.

3.3 Local and Global Constraints

3.3.1 Constraints as Cost functions

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 problem divided by the size of the
sequence.
```

More complex constraints can be defined as easily. For instance,

- distance constraint : forces each variables V_i in scope to have values X_i for which a given numerical metadata $p(X_i)$ is as close as possible as a target value p_t (e.g. "all these variables should have an energy of 0.1"). The corresponding cost function is defined as follows :

  `DistanceCt.cost()`
  `Return the mean distance between the` $p(X_i)$ `and` p_t`, i.e.` $\frac{1}{M}\sum_{i=0}^{M-1}|p(X_i) - p_t|^2$
- continuity constraint : holds on a set of variable $V_i, i = 1..s$. It forces each duplet of successive variables $\{V_i, V_{i+1}\}$ to have values $\{X_i, X_{i+1}\}$ for which a given numerical metadata p has similar values $\{p(X_i), p(X_{i+1})\}$. The corresponding cost function can be defined as follows :

  `ContinuityCt.cost()`
  `Return the mean distance between all` $p(X_i)$ `and` $p(X_{i+1})$`, i.e.` $\frac{1}{M}\sum_{i=0}^{M-2}|p(X_i) - p(X_{i+1})|^2$

In practice, the cost functions are implemented more efficiently, by passing as argument the lastly modified variable to the cost functions (which in our context, is always the current variable V_n). This information is used to compute only the differential cost, instead of the whole cost.

For instance, the cost function of the distance constraint can be defined in such a differential way :
`DistanceCt.cost(Variable v)`
`Returns` $\frac{1}{M}(oldcost * (M-1) + |p(X_n) - p_t|^2)$
This saves $M-1$ database accesses to compute the $p(X_i)$, and $M-1$ substractions and multiplications. Such optimizations are important, as the cost function of each constraint is called N times at each iteration, where N is the size of the current variable's domain.

3.3.2 Local and Global Constraints

In the context of incremental CSP, and for the clarity of further discussions in section 4.2, we distinguish 2 types of constraints :

- Local Constraints only hold on the current variable. They influence the selection of the next drumbar by only looking at its intrinsic properties, without taking past values into account. Typically, a distance constraint is a local constraint.
- Global Constraints hold on the current variable plus some or all of the past variables. They influence the selection of the best drumbar by also accounting for the values of the past variables. Typically, a continuity constraint is a global constraint, trying to select new values so that they are continuous with the past, already selected values.

Upon increment of the CSP, local and global constraints have a different behavior. All local constraints update their scope by removing the previous current variable (now v_{n-1}), and adding the new current variable v_n. All global constraints simply add the newly added variable to their scope. This mechanism is illustrated in Figures 3 and 4: before the increment, the global constraints G_1 and G_2 hold on $\{V_1, V_2, V_3\}$ and the local constraints L_1 and L_2 hold only on V_3, which is the current variable. After increment, G_1 and G_2 modify their scope to also include the

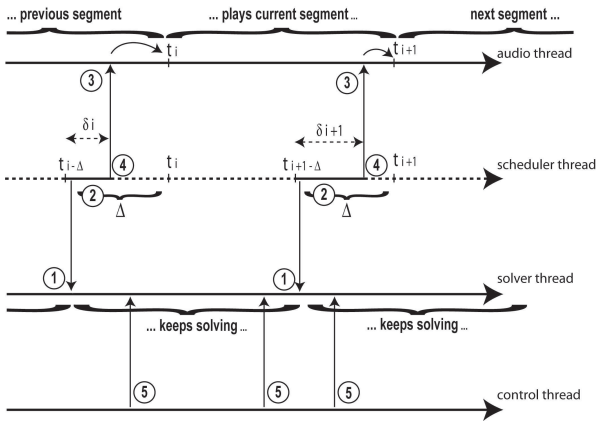

Figure 5: The real-time implementation of the system, using 4 concurrent threads. See main text for explanation of steps 1 to 5.

new current variable V_4, while L_1 and L_2 now only hold on V_4.

3.4 Real-time Implementation

The real-time implementation of the system (done in Java) uses several concurrent threads. A *solver* thread is given a CSP, and solves it using the algorithm described in section 3.2. A *audio* thread is in charge of the continuous playback of the drumtrack, by concatenating the values of the successive current variables. A *scheduler* thread iteratively queries the solver for the best solution so far and schedules the corresponding audio for playback by the audio thread. Finally, a *control* thread can modify at any time the constraints holding on the CSP which the solver is currently working on.

Figure 5 explicits the interactions between the 4 threads. At any time, the audio corresponding to the latest selected drumbar is playing, and a new value must be scheduled to immediately start after it finishes at endtime t_i.

1. At time $t_i - \Delta$, the scheduler wakes up, and asks the solver for the best value it has found so far for the current variable, given all the current constraints and the values of the past variables. The solver replies and increments immediately to start looking for the next drumbar.

2. The scheduler retrieves the audio corresponding to the drumbar found at step1. In the current implementation, this includes reading and decoding a .mp3 file between a start and end date through a local network, which may take a variable time δ_i.

3. At time $t_i - \Delta + \delta_i$, the scheduler thread schedules the decoded audio for the current drumbar for playback at the exact ending time t_i of the latest drumbar, which is currently being played. The choice of Δ is made a priori, to ensure that $t_i - \Delta + \delta_i < t_i$, i.e. $\Delta > \delta_i, \forall i$. In our current implementation, we chose $\Delta = 500ms$.

4. The scheduler sleeps until $t_{i+1} - \Delta$, having $t_{i+1} = t_i + d_i$, where d_i is the duration of the audio just scheduled. This mainly gives the priority back to the solver, which keeps scouring the database for the next value to be scheduled at t_{i+1}.

5. At any time, the control thread may modify the constraints holding on the CSP. This in turn modifies the values found by the solver, which enables the real-time high-level control of the drummer's output. Such changes can be done manually by a user via a GUI, or result of the analysis of an interaction, as proposed below in Section 4.1.

4 Experiments

4.1 Midi interaction

We describe here preliminary experiments in which we use the drummer agent in interaction with a human player playing a midi keyboard. We automatically build a database of 150 drumbars extracted from a set of demo songs recorded with BFD (BFD, FXpansion Audio UK Ltd, 2003), and automatically compute the associated metadata (see Section 2.4). The drumtrack is constrained to use only drum samples with a bpm of 120, so that the resulting concatenation has a steady beat.

The midi performance of the human player is analysed in real-time to extract to following information :

- energy : the mean velocity of the `note-on` Midi messages, computed over 500 ms windows (value $\in [0, 127]$).
- onset density : the ratio number of `note-on` Midi messages received over 500 ms windows, to a practical maximum of 10 notes[3] (value $\in [0, 1]$).
- pitch : the mean midi pitch of the `note-on` Midi messages, computed over 500 ms windows (value $\in [0, 1]$).

The three streams of midi metadata are converted using a transfer function, and sent to the drummer which modifies its local constraint set accordingly. The drummer thus generates a drumtrack by satisfying constraints created by analysing the Midi performance. For instance, a new Midi energy value modifies a local distance constraint holding on the drumbars' energy metadata, i.e. which forces the energy of the newly selected drumbars to be as close as possible to the input midi energy. Similarly, low midi pitch can be inverse converted, and mapped to a local constraint holding on the presence of cymbal metadata, so that melodies played on the lower octaves of the midi instrument trigger drum track that use a lot of high pitched sounds, and conversely, high pitched melodies trigger a lot of bass drum and tom sounds. Mappings between midi performance data and audio drumtrack metadata/constraints can be arbitrary complex. In the current

[3] corresponding to one note every 50 ms. Pachet (2002) analysed phrases played by John McLaughlin said to be one of the fastest jazz guitarists and found a minimum inter onset time of about 60 milliseconds, so this works as a practical lower bound.

system, the mappings are hardcoded, but they could be modified in real-time. This issue of mapping between Midi performance and Machine generated music parameters is discussed at length in Rowe (1993).

Figure 6 shows both the energy of the midi performance and the energy of the drumtrack generated by the drummer over time. We observe that the drummer is able to follow the energy of the performance very finely, with very small latency (typically the duration of one drumbar), and a precision which depends on the available energy values in the database.

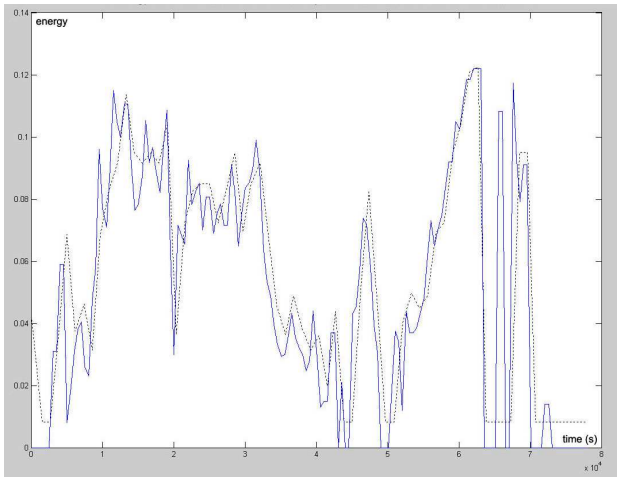

Figure 6: Energy of the Midi performance (solid line) and energy of the drumbars returned by the solver (dashed line) over time.

4.2 Autonomy/Reactivity Trade-off

While technically satisfying, such fully reactive behaviour is often not suitable in a music interaction context, in which one wants the interacting agent to have both musical realism and autonomy. For instance, it may be unrealistic to instantly switch from very low to very high energy. In our system, we use global constraints to counterbalance the immediate reactivity created by the local constraints. Consider 2 constraints holding on the drumbars' energy:

- a local DistanceConstraint which forces the drumtrack's energy to match the midi performance's energy (as above)
- a global ContinuityConstraint, which forces the consecutive selected drumbars to have similar energy

These 2 constraints are contradictory: if the performance energy suddenly increases, highly energetic drumbars will have a low cost according to the local distance constraint, but a high cost according to the global continuity constraint. On can manipulate the total cost of a drumbar x_i by putting weights on the local and global constraints:

$$cost(x_i) = \alpha.cost_{local}(x_i) + \beta.cost_{global}(x_i) \qquad (5)$$

and a variety of behaviours can be achieved ranging between complete reactivity (0-weight on the global constraint) and complete autonomy (0-weight on the local constraint). Figure 7 shows the behaviour of the drummer subjected to a typical Midi energy input, using different ratio between local weight and global weight. With intermediate settings, the drummer follows the input energy while still preserving continuity, thus yielding a more musical output.

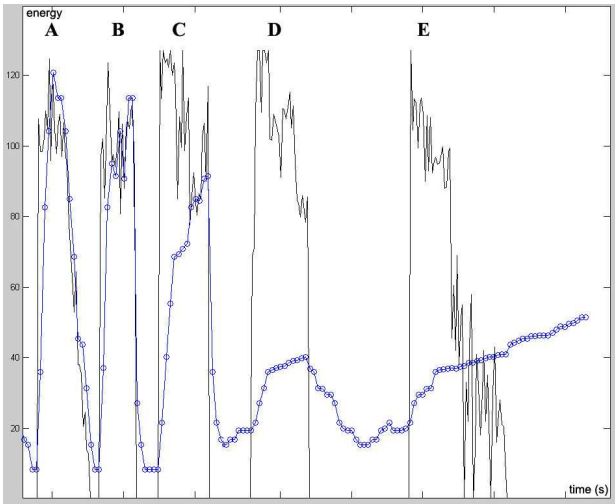

Figure 7: Drummer behaviour with different ratio r of local to global weight. (A) $r = \infty$: the drummer reacts immediately. (B) $r = 2$. (C) $r = 1$. (D) $r = \frac{1}{2}$. (E) $r = 0$: complete autonomy of the drummer.

5 Conclusion and Future Work

We described a real-time drummer inspired by Music Information Retrieval techniques. It generates an audio drum-track by concatenating drum segments automatically extracted from pre-existing musical files. The drumtrack can be controlled in real-time by local and global constraints holding on the metadata of the drumbars, using a custom constraint satisfaction algorithm based on local search. We designed several drum track audio descriptors: drum detection, energy, onset density, presence of drums and presence of cymbals. We also reported on simple experiments made in the context of a virtual duet between the system and a human pianist, and show that the weights between local and global constraints can be manipulated to control the autonomy/reactivity of the system.

This paper describes a basic mecanism of competitive local/global constraint satisfaction, which can be extended to support additional metadata, more refined constraints, and more complex interaction mappings. Audio drumbars can be automatically indexed with more advanced rhythmic descriptors, such as syncopation, strong/weak beats, or style (e.g. for jazz drumming : swing, bebop, latin, ethnic, free, etc.). More complex autonomy behaviours can be achieved with additional global constraints, such as distribution constraints (e.g. "60% of the drumbars should use a latin pattern"), or sequence constraints (e.g. "a fill with a lot of toms every 4 drumbars"). Additional information can be extracted from the Midi performance (notably metric analysis to match the audio). Finally, con-

straints and mappings can also be changed in real-time based on the midi analysis.

6 Acknowledgement

This work uses JSyn, a Java synthesis library (Burk, 1998), and MidiShare, a real-time multi-task MIDI operating system developed by GRAME (Orlarey and Lequay, 1989). It has been partially founded by The SemanticHifi European IST project.

References

Bfd, premium acoustic drum library module, FXpansion Audio UK Ltd, 2003. website: http://www.fxpansion.com.

Dfh, drumkit from hell, 2003. Toontrack Music, website: http://www.toontrack.com.

Jean-Julien Aucouturier and François Pachet. Scaling up music playlist generation systems. In *Proceedings of The IEEE International Conference on Multimedia and Expo, Lausanne (Switzerland)*, August 2002.

Jean-Julien Aucouturier, François Pachet, and Peter Hanappe. From sound sampling to song sampling. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spain., 2004.

Jeff A. Bilmes. Techniques to foster drum machine expressivity. In *Proceedings of the International Computer Music Conference*, 1993.

Phil Burk. Jsyn, a real-time synthesis api for java. In *Proceedings of the International Computer Music Conference (ICMC)*. ICMA, 1998. available: http://www.softsynth.com/jsyn/.

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.

Perfecto Herrera, Alexandre Yeterian, and Fabien Gouyon. Automatic classification of drum sounds: a comparison of feature selection methods and classification techniques. In *Proceedings of Second International Conference on Music and Artificial Intelligence*, Edinburgh, Scotland, 2002.

J. Koza. *Genetic Programming*. MIT Press., 1992.

Ari Lazier and Perry Cook. Mosievius: Feature-driven interactive audio mosaicing. In *Proceedings of the COST-G6 Conference on Digital Audio*, London, UK, September 2003.

Y. Orlarey and H. Lequay. Midishare: a real time multitasks software module for midi applications. In *Proceedings of the 1989 International Computer Music Conference, San Francisco*. Computer Music Association, 1989.

F. Pachet. Music interaction with style. In *Proceedings of the International Computer Music Conference*, 2002.

François Pachet. Rhythm as emerging structure. In *Proceedings of the International Computer Music Conference*, Berlin, Germany, 2000.

François Pachet and Olivier Delerue. On-the-fly multitrack mixing. In *Proceedings of AES 109th Convention*, Los Angeles, USA., 2000.

François Pachet and Pierre Roy. Musical harmonization with constraints: A survey. *Constraints*, 6(1):7–19, 2001.

J. Paulus and A. Klapuri. Measuring the similarity of rhythmic patterns. In *Proceedings, 3rd International Conference on Music Information Retrieval*, Paris, France, October 2002.

Robert Rowe. *Interactive Music Systems*. MIT Press, Cambridge, Massachusetts, 1993.

Eric Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustic Society of America*, 103(1):588–601, January 1998.

Eric Scheirer and Malcolm Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *Proceedings of ICASSP*, 1997.

Diemo Schwarz. The caterpillar system for data-driven concatenative sound synthesis. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK., September 2003.

Nao Tokui and Hitoshi Iba. Music composition by means of interactive ga and gp. In *Proceedings of IEEE Systems, Man and Cybernetics*, 2001.

Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G. Okuno. Automatic drum sound description for real-world music using template adaptation and matching methods. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spain, 2004.

M. Zaanen, R. Van Bod, and H. Honing. A memory-based approach to meter induction. In *Proceedings of the ESCOM*, pages 250–253., 2003.

Aymeric Zils and François Pachet. Musical mosaicing. In *Proceedings of the COST-G6 Conference on Digital Audio*, Limerick, Ireland, December 2001.

Aymeric Zils and François Pachet. Extracting automatically the perceived intensity of music titles. In *Proceedings of the 6th COST-G6 Conference on Digital Audio Effects (DAFX03)*, London, UK, September 2003.

Aymeric Zils and François Pachet. Automatic extraction of music descriptors from acoustic signals using eds. In *Proceedings of the 116th AES Convention, Berlin*, May 2004.

Aymeric Zils, François Pachet, Olivier Delerue, and Fabien Gouyon. Automatic extraction of drum tracks from polyphonic music signals. In *Proceedings of Web Delivery of Music (WEDELMUSIC)*, December 2002.