REAL-TIME CORPUS-BASED CONCATENATIVE SYNTHESIS FOR SYMBOLIC NOTATION

Daniele Ghisi
STMS Lab
(IRCAM, CNRS, UPMC)
danieleghisi@bachproject.net

Carlos Agon
STMS Lab
(IRCAM, CNRS, UPMC)
carlos.agon@ircam.fr

ABSTRACT

We introduce a collection of modules designed to segment, analyze, display and sequence symbolic scores in real-time. This mechanism, inspired from *CataRT*'s corpus-based concatenative synthesis, is implemented as a part of the *dada* library for Max, currently under development.

1. INTRODUCTION

Corpus-based concatenative synthesis is a largely known technique, providing mechanisms for real-time sequencing of grains (extracted from a large corpus of segmented and descriptors-analyzed sounds), according to their proximity in some descriptors space. Among the existing tools dealing with such technique, the *CataRT* modules [1] are probably the most widely used: taking advantage of the features in the FTM library [2], they allow segmentation and analysis of the original sound files, as well as the exploration of the generated corpus via an interactive two-dimensional display, inside the Max environment.

CataRT is oriented to real-time interaction on audio data, essentially omitting any symbolic representation of events. Although some work has been done recently to link symbolic notation with *CataRT*, such as [3], none of these works, to the best of our knowledge, is meant to fully bring the ideas of concatenative synthesis into the symbolic domain.

In this article we describe a corpus-based concatenative system designed and implemented in order to bring into Max the ability to segment, analyze and explore symbolic scores, in a similar fashion than *CataRT* does with sounds. This system will be distributed as part of the *dada* library (currently under development [1]), which will contain a set of non-standard two-dimensional interfaces dealing with symbolic musical content. The *dada* library, in turn, is based upon the *bach* library, which provides Max with a set of tools for the graphical representation of musical notation, and for the manipulation of musical scores through a variety of approaches ranging from GUI interaction to constraint programming, and sequencing. The *bach* library is oriented to real-time interaction, and is meant to interoperate easily with other processes or devices controlled by Max, such as DSP tools, MIDI instruments, or generic hardware systems [4, 5]. A number of high-level modules based on *bach*, solving typical algorithmic and computer-aided composition problems, have also been collected to form the *cage* library [7].

The system we describe in this article naturally extends the concept of score granulation (introduced in [6] and then later implemented in the *cage.granulate* module [7]), allowing a finer control on the concatenation of grains, according to some relationships between the grain features extracted during the analysis process. Moreover, the feature extraction is heavily based on the *bach* lambda loop visual programming pattern [5], hence making analysis fully customizable.

2. OVERVIEW AND MODULES

The system relies on three different modules: *dada.segment*, performing segmentation and feature extraction, *dada.base*, implementing the actual database engine, and *dada.catart*, a two-dimensional graphic interface capable of organizing and interacting with the extracted grains.

2.1 Segmentation

The *dada.segment* module performs the segmentation of the original scores, contained either in a *bach.roll* (as unmeasured musical data) or *bach.score* (as classically notated musical data), in one of the following manners:

- Via markers: each marker in the original *bach.roll* is considered as a cutting point at which the score is sliced. All the slices (grains) are then collected.
- Via equations: a single value (in milliseconds for *bach.roll*, or as a fraction of the bar or beat duration, for *bach.score*) or more generally an equation can be used to establish the size of each grain. In *bach.roll* this equation can take as variable the grain onset, and is especially useful when segmentation needs to

[1] A 0.0.1 alpha version of *dada* is publicly available at the address http://data.bachproject.net/download.php?file=dada_0.0.1.zip), requiring Max 6.1.7 or higher (http://cycling74.com), bach 0.7.8.5 or higher (http://data.bachproject.net/download.php?file=bach_0.7.8.5.zip), and cage 0.3.5 or higher (http://data.bachproject.net/download.php?file=cage_0.3.5.zip). This release is actually a very crude Macintosh-only release, including the very first modules of *dada*. Among such modules are all the tools used within the scope of this paper. A modified version of the two examples proposed in section 3 can be accessed via *dada.catart*'s help file.

be performed roughly independently from the musical content itself. In *bach.score*, voices are presegmented into chunks of measures (according to a pattern established via the 'presegment' attribute), and each chunk is in turn segmented into grains whose duration is determined by the aforementioned equation - possibly having as variables the measure number, the measure division (beat), and the measure overall symbolic duration (see for instance fig. 1).

- Via label families: differently from sound files, scores easily allow non-vertical segmentations, where only a portion of the musical content happening in a given time span is accounted for (see fig. 2). If labels are assigned to notes or chords in the original score, a grain is created for each label, containing all the elements carrying such label.

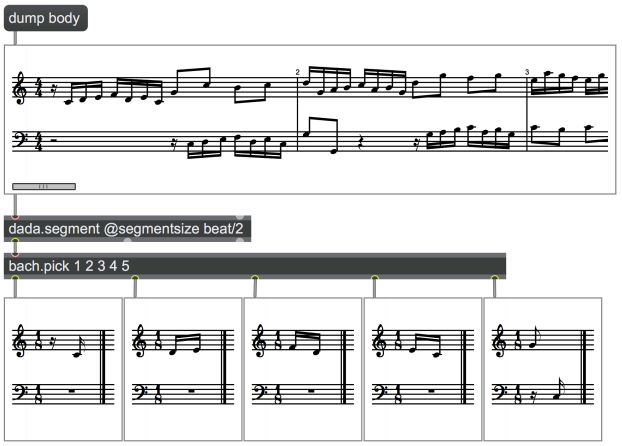

Figure 1. Segmentation of a *bach.score* into grains having length equal to half of the beat (i.e. an eighth note). The first five grains are displayed in the bottom part of the patch.

2.2 Analysis

Grain analysis is performed during the segmentation process. On one side, *dada.segment* is capable of adding some straightforward metadata to the segmented grains, such as their duration, onset, index, label (if segmentation is carried out via label families) and notation object type (either 'roll' for *bach.roll* or 'score' for *bach.score*); in case the grain comes from a *bach.score*, tempo, beat phase (the beat on which the grain starts), symbolic duration and bar number can also be added.

On the other hand, *dada.segment* allows the definition of custom features via a lambda loop mechanism (see [3, 5]): grains to be analyzed are output one by one from the rightmost (lambda) outlet, preceded by the custom feature name; the user should provide a patching algorithm to extract the requested feature, and then plug the result back into *dada.segment*'s rightmost (lambda) inlet. Feature names, defined in an attribute, are hence empty skeletons which will be 'filled' by the analysis implementation, via patching. This programming pattern is widely used

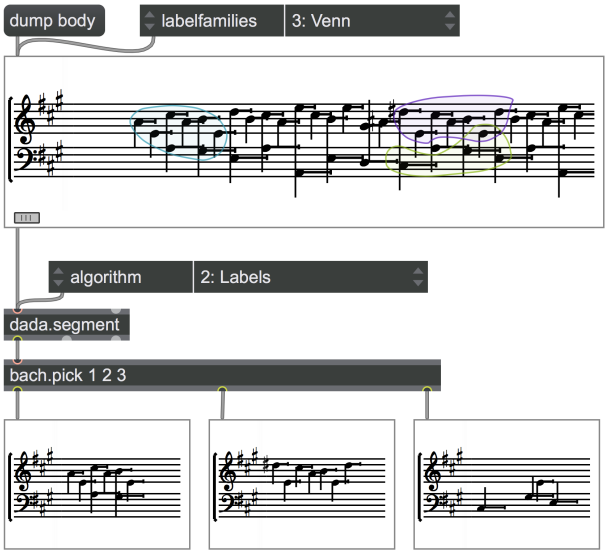

Figure 2. Segmentation of a *bach.roll* according to label families. Labeled items are automatically enclosed in colored contours in the original *bach.roll*. Notice how families can overlap (in the example above, one note is labeled twice, and hence assigned to two families at the same time). The first three grains (corresponding to the first three label families) are displayed in the bottom part of the patch.

throughout the *bach* library (one can easily compare the described mechanism, for instance, with *bach.constraints*'s way of implementing custom constraints [5]), and allows users to implement virtually any type of analysis on the incoming data. Nevertheless, some ready-to-use abstractions are provided (see fig. 3) for standard features such as centroid, spread, loudness or item counting.

Analyzed features are collected for each grain, and output as metadata from the middle outlet of *dada.segment*.

2.3 Database

Once the score grains have been produced and analyzed, they are stored in a SQLite database, whose engine is implemented by the *dada.base* object. Hence, data coming from *dada.segment* are properly formatted and fed to *dada.base*, on which standard SQLite queries can be performed (see figure 3). Databases can be saved to disk and loaded from disk.

2.4 Interface

Finally, the *dada.catart* object provides a two-dimensional graphic interface for the database content. Its name is an explicit acknowledgment to the piece of software which inspired it. Grains are by default represented by small circles in a two dimensional plane. Two feature can be assigned to the horizontal and vertical axis respectively; two more features can be mapped on the color and size of the circles. Finally, one additional integer valued feature can be mapped on the grain shape (circle, triangle, square, pentagon, and so forth), adding up to a total number of five

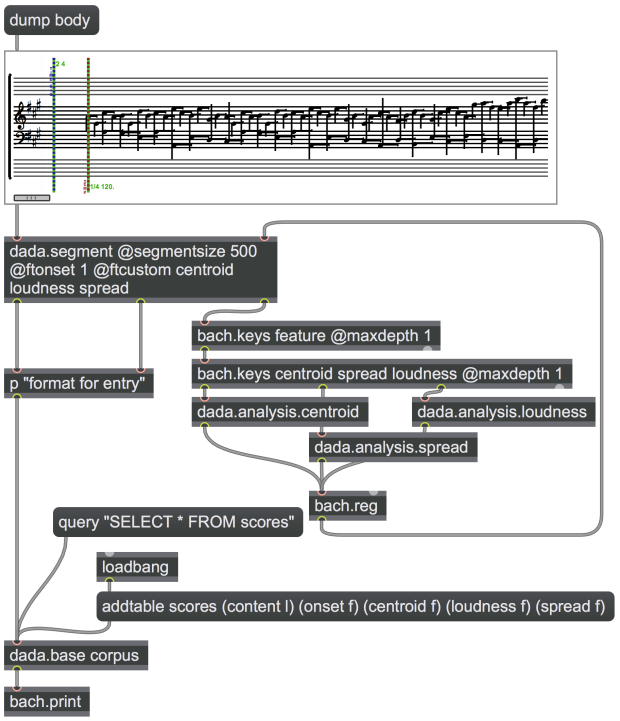

Figure 3. When the patch opens, a table named 'scores' is created in the database named 'corpus', collecting all the grains. This table has five columns: the content of the grain (a *bach* Lisp-like linked list), the onset the grain had in the original score, its centroid, loudness and spread (all floating point numbers). When the 'dump body' message is clicked, the score contained in the *bach.roll* is segmented and analyzed by centroid, loudness and spread (respectively computed via the *dada.analysis.centroid*, *dada.analysis.spread* and *dada.analysis.loudness* modules inside the lambda loop). The database is then filled, and standard SQLite queries can be performed on it.

features being displayed at once (see fig. 4). The database elements can be sieved by setting a *where* attribute, implementing a standard SQLite 'WHERE' clause. The vast majority of the display features can be customized, such as colors, text fonts, zoom and so on.

Each grain is associated with a 'content' field, which is output either on mouse hovering or on mouse clicking. The content is usually assigned to the *bach* Lisp-like linked list representing the score [5]. The sequencing can also be beat-synchronous, provided that a tempo and a beat phase fields are assigned: in this case the content of each grain is not output as soon as the grain is clicked upon (or mouse hovered), and its sequencing is postponed in order for it to align with the following beat, according to the current tempo (obtained from the previously played grains).

In combination with standard patching techniques, these features also allow the real-time display, sequencing and recording of grains (see section 3 for an example).

A *knn* message allows to retrieve the k-th nearest samples for any given (x, y) position. A system of messages inspired by turtle-graphics is also implemented, in order to be able to move programmatically across the grains;

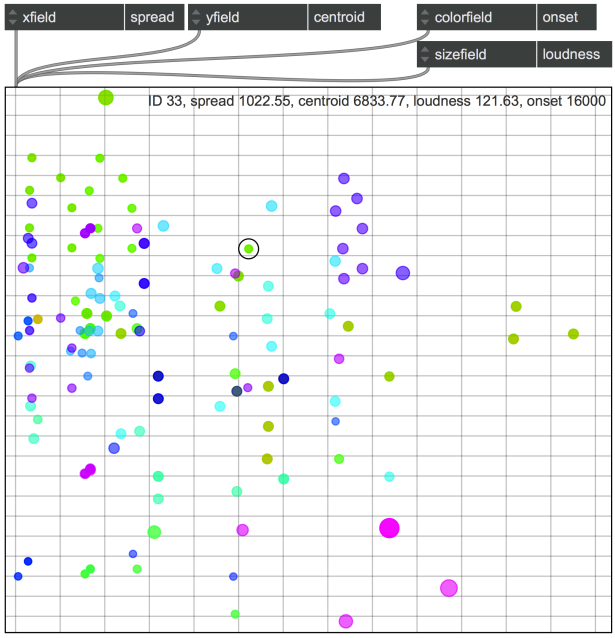

Figure 4. The *dada.catart* object displaying the database built in figure 3. Each element of the database (grain) is represented by a circle. On the horizontal axis grains are sorted according to the spread, while on the vertical axis grains are organized according to their centroid. The colors scale is mapped on the grain onsets, while the circle size represents the grain loudness.

namely a 'turtle' grain can be assigned via the *setturtle* message (setting the turtle on the nearest grain with respect to a given (x, y) position), and then the turtle can range across the grains via the *turtle* message, moving it of some $(\Delta x, \Delta y)$ and then choosing the nearest grain with respect to the new position (disregarding the original grain itself). The turtle is always identified in a *dada.catart* by an hexagon (see fig. 7 for an example).

3. EXAMPLES

3.1 An interactive tonal centroid palette

As a first example, in the patch displayed in figure 6 we segment (in grains of 1 second each) and then analyze the first eight Lieder from Schubert's *Winterreise*. During the analysis process we take advantage of the tonal centroid transform proposed by Harte, Sandler and Gasser in [8], and implemented in the *cage* library (see [7]). The horizontal axis displays the phase of the tonal centroid with respect to the plane of fifths, while the vertical axis displays the phase referred to the plane of minor thirds (both range from -180 to 180 degrees). The analysis subpatch computing the phase of the projection of the tonal centroid on the plane of fifths is displayed in fig. 5 (the one for minor thirds is analogous). Both colors and shapes are mapped on the Lieder number.

We can use this representation as a sort of 'interactive tonal centroid palette': each vertical line refers to a note in the circle of fifths, each horizontal line refers to an aug-

mented chord in the circle of minor thirds. If we focus especially on the horizontal axis, we notice for instance that red circles (belonging to the first Lied, *Gute Nacht*, in D minor) are mostly scattered around the vertical line referring to the D, or that orange triangles (belonging to the second Lied, *Die Wetterfahne*, in A minor) are mostly scattered in the region around A.

A record mechanism is implemented, and the recorded data is collected in the score displayed at the bottom of the image. The score can then be saved, quantized or exported, taking advantage of the features of the *bach* library [5].

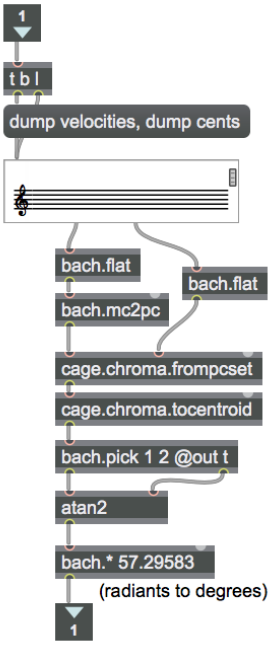

Figure 5. The subpatch computing the phase of the projection of the tonal centroid of a *bach.roll* grain on the plain of fifths. All the pitches, converted to pitch classes, are weighted with their own velocities and gathered in a chroma vector, whose tonal centroid is computed via the algorithm proposed in [8]. The first two components of the tonal centroid (referring to the plane of fifths) are picked, and the angle formed by the vector is computed.

3.2 Rearranging beats

As a second example, consider figure 7, where we have segmented the first Bach invention (BWV 772) by beat. On horizontal axis we display the measure number, on vertical axis we display the position of the beginning of the grain inside the measure (phase). We can then send *turtle* messages in order to navigate through the grains, so that we can read the complete score as it was (patch mechanism at top left corner of the image), or only read the last beats of each measure from last to first measure (top-middle part of the image), or even move in random walks among the beats (top-right part of the image).

4. CONCLUSION AND FUTURE WORK

We have presented a system operating on symbolic musical content, directly inspired by the *CataRT* modules, and implemented as part of the *dada* library for Max, currently under development. This system naturally extends the symbolic granulation engine implemented in *cage.granulate*, allowing to organize score grains according to custom analysis features.

This system can be improved in a certain number of ways. For one thing, the number of predefined analysis modules should be increased, by bridging into the symbolic domain important audio descriptors such as roughness, inharmonicity, and so on. The relationships between audio and symbolic descriptors could be in itself a topic for further investigations. Moreover, the *dada.segment* module is currently able to segment based on given markers, equations or labels; however it is not able, by design, to infer such markers or labels. One of the interesting topics of future research might hence be to integrate inside the process a system for semi-automatic segmentation of scores, and a module for pattern retrieval. Also, the label-based extraction currently works only for *bach.roll*, and a *bach.score* version of such an algorithm should be also implemented.

5. REFERENCES

[1] D. Schwarz, G. Beller, B. Verbrugghe, and S. Britton, "Real-time corpus-based concatenative synthesis with catart," in *IN PROC. OF THE INT. CONF. ON DIGITAL AUDIO EFFECTS (DAFX-06*, 2006, pp. 279–282.

[2] N. Schnell, R. Borghesi, D. Schwarz, F. Bevilacqua, and R. Müller, "FTM - Complex Data Structures for Max," in *Proceedings of the International Computer Music Conference*, 2005.

[3] A. Einbond, C. Trapani, A. Agostini, D. Ghisi, and D. Schwarz, "Fine-tuned control of concatenative synthesis with catart using the bach library for max," in *Proceedings of the International Computer Music Conference*, Athens, Greece, 2014.

[4] A. Agostini and D. Ghisi, "Real-time computer-aided composition with *bach*," *Contemporary Music Review*, no. 32 (1), pp. 41–48, 2013.

[5] ——, "A max library for musical notation and computer-aided composition," *Computer Music Journal*, vol. 39, no. 2, pp. 11–27, 2015/10/03 2015. [Online]. Available: http://dx.doi.org/10.1162/COMJ_a_00296

[6] ——, "*bach*: an environment for computer-aided composition in Max," in *Proceedings of the International Computer Music Conference (ICMC 2012)*, Ljubljana, Slovenia, 2012, pp. 373–378.

[7] A. Agostini, E. Daubresse, and D. Ghisi, "*cage*: a High-Level Library for Real-Time Computer-Aided Composition," in *Proceedings of the International Computer Music Conference*, Athens, Greece, 2014.

[8] C. Harte, M. Sandler, and M. Gasser, "Detecting harmonic change in musical audio," in *In Proceedings of Audio and Music Computing for Multimedia Workshop*, 2006.

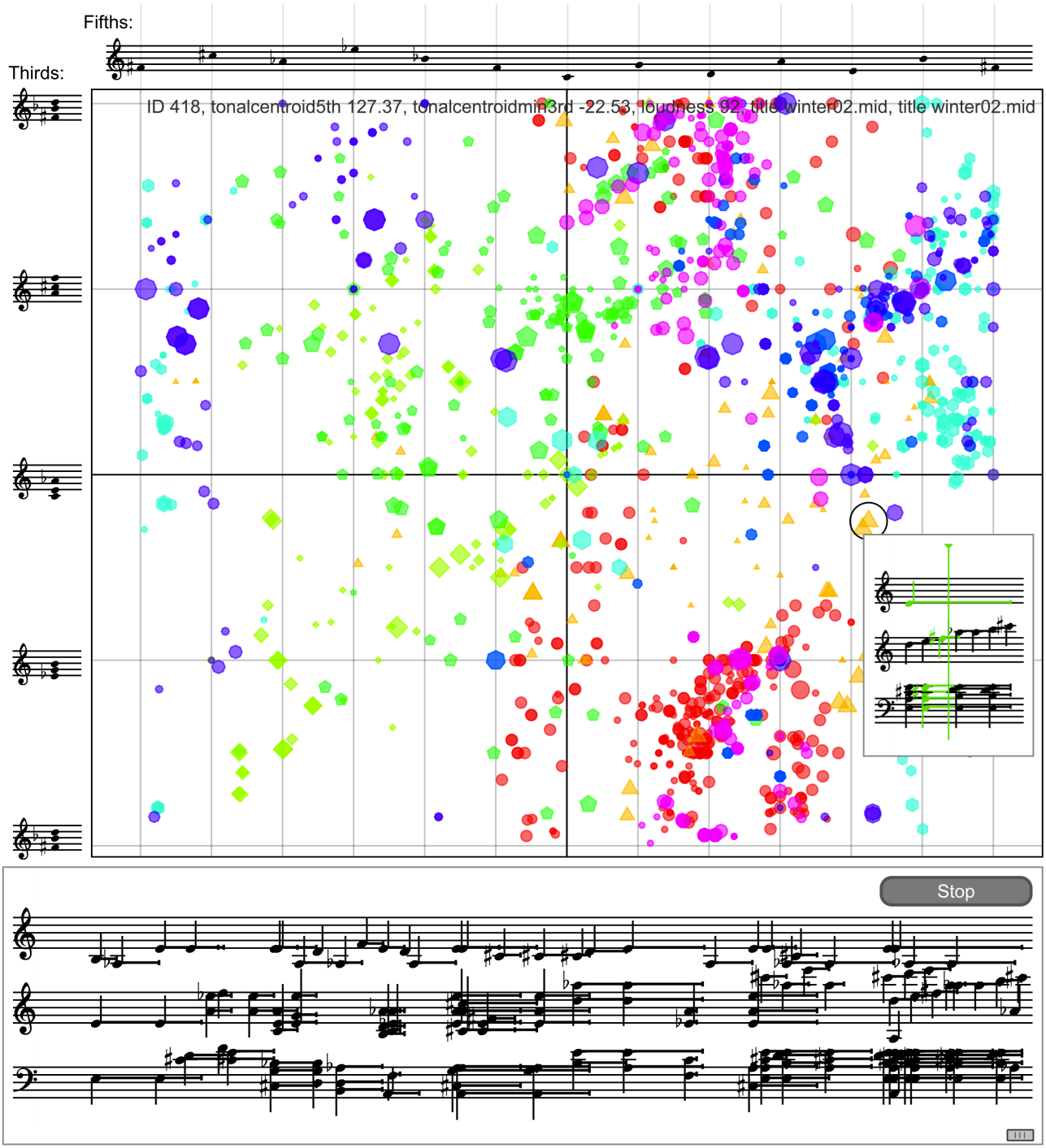

Figure 6. A patch displaying the database build from the first eight Lieder of Schubert's *Winterreise*, organized by tonal centroids (the phase of the projection on the plane of fifths is on horizontal axis, the phase of the projection on the plane of minor thirds is on the vertical axis). Both colors and shapes identify the Lieder number (1 being the circle, 2 being the triangle, 3 being the square, and so on). When the recording mechanism is turned on, grains can be played via mouse hovering, and the bottommost *bach.roll* contains the recorded result.

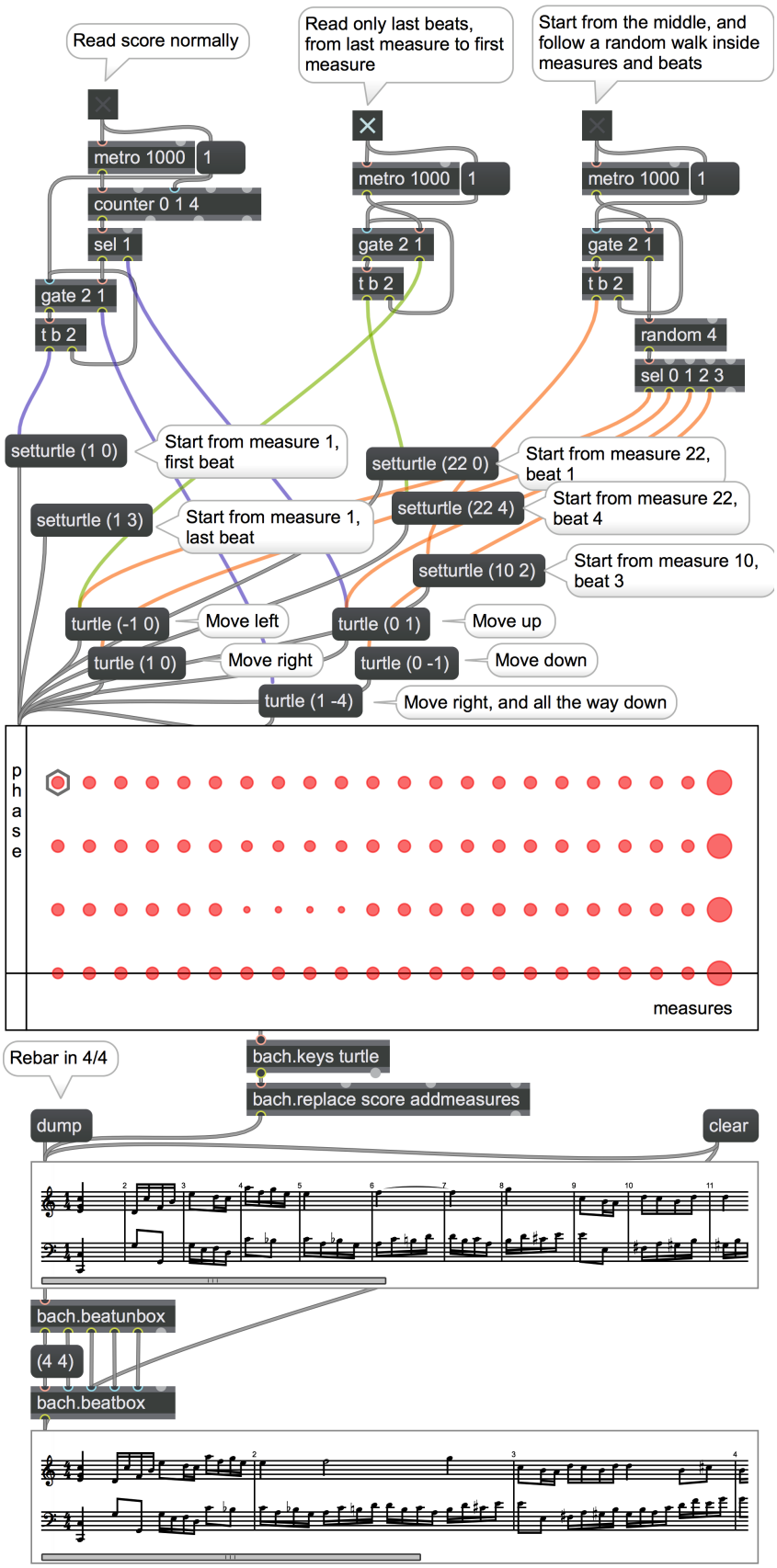

Figure 7. An example showing the manipulation of the first Bach invention (BWV 772), segmented by beat, and rearranged so to play and record the last beats of each measure (starting from last measure, and ending with first one). Notice how ties are preserved during the segmentation process (e.g. between measure 6 and 7) of the upper *bach.score*, rebarred in measure 2 of the lower one.