# A Supervised Approach for Rhythm Transcription Based on Tree Series Enumeration

**Adrien Ycart**
Sorbonne Universités
STMS (IRCAM-CNRS-UPMC)
Paris, France
adrien.ycart@ircam.fr

**Florent Jacquemard**
INRIA – Sorbonne Universités
STMS (IRCAM-CNRS-UPMC)
Paris, France
florent.jacquemard@inria.fr

**Jean Bresson**
Sorbonne Universités
STMS (IRCAM-CNRS-UPMC)
Paris, France.
jean.bresson@ircam.fr

**Sławek Staworko**
University of Edinburgh
Scotland
slawomir.staworko@inria.fr

## ABSTRACT

*We present a rhythm transcription system integrated in the computer-assisted composition environment OpenMusic. Rhythm transcription consists in translating a series of dated events into traditional music notation's pulsed and structured representation. As transcription is equivocal, our system favors interactions with the user to reach a satisfactory compromise between various criteria, in particular the precision of the transcription and the readability of the output score. It is based on a uniform approach, using a hierarchical representation of duration notation in the form of rhythm trees, and an efficient dynamic-programming algorithm that lazily evaluates the transcription solutions. It is run through a dedicated user interface allowing to interactively explore the solution set, visualize the solutions and locally edit them.*

## 1. INTRODUCTION

We call rhythm transcription the act of converting a temporal stream such as the onsets in a sequence of notes into a musical score in Western notation. The note series can come from a musician's performance, or can be generated by an algorithm, for instance in a computer-assisted composition (CAC) environment such as OpenMusic [1]. In this article, we will particularly focus on the latter case.

Rhythm transcription is a long-discussed computer-music challenge [2], which can be divided into several sub-tasks (beat tracking, tempo/meter estimation, etc.) that are often considered as Music Information Retrieval (MIR) problems on their own [3, 4].

In traditional music notation, durations are expressed as fractions of a unit (*beat*) given by the *tempo*. The durations in physical time (in seconds) thus need to be converted in musical time, which requires a tempo (in beats per minute) to be inferred. The duration values have to belong to the small set defined by successive divisions of the beat (eighth notes, sixteenth notes, *etc*). The input durations thus have to be approximated (once converted into musical time) by admissible note values. We call this task *rhythm quantization*. Transcription can also be made easier by a first *segmentation* step, cutting the input stream into smaller units (if possible, of constant tempo) that are easier to analyze.

One of the difficulties of rhythm transcription stems from the coupling between tempo estimation and quantization. On the one hand, the durations cannot be quantized without knowing the tempo, and on the other hand, the quality of the transcription can only be assessed after obtaining the result of quantization. The situation is thus a chicken-and-egg problem [5].

Apart from that problem, rhythm quantization itself is difficult as the solution is not unequivocal: for a given input series of notes, several notations are admissible, and they can be ranked according to many different criteria. One of the criteria is the *precision* of the approximation, *i.e.* how close the output is to the input in terms of timing. Another important criterion is the *complexity* of the notation, *i.e.* how easy it is to read. These two criteria are often contradictory (cf. figure 1) : in general, the more precise the notation is, the more difficult it is to read. Thus, to yield good results, quantization must be a compromise between various criteria.

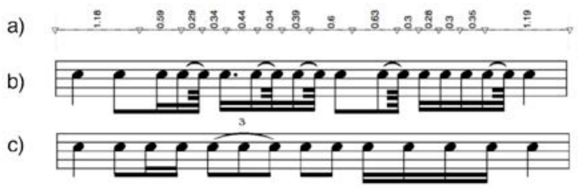


**Figure 1**. Figure taken from [6] : a) Input sequence. b) A precise but complex notation of the input sequence. c) Another notation of the same sequence, less precise but less complex.

Moreover, the same series of durations can be represented by various note values, as shown in Figure 2. Even if they represent the same durations, some of those transcriptions can be more readable than others. They can also have different musical meanings, and be interpreted differently.

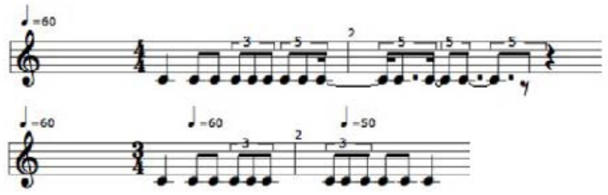


**Figure 2**. Two equivalent notations of the same series of durations, the second one being more readable.

All these ambiguities make rhythm quantization a hard problem, and developing an fully automatic system that compromises between precision and complexity, all the while respecting, in the case of CAC, the message the composer wants to convey, is not realistic. Moreover, a single-solution approach, returning the *optimal* transcription may be unsatisfactory in many cases. Indeed, there is no ideal compromise between the criteria.

In this article, we present a multi-criteria enumeration approach to transcription, integrated in OpenMusic. Our aim is to enumerate the various possible transcriptions, from best to worst, according to a given set of criteria. This approach is different from a single-solution approach, as we study supervised, interactive frameworks, where the user guides the algorithm throughout the process to converge to a solution. Besides, our approach allows an original coupling of the tempo estimation and quantization tasks.

The first step is the construction of a structure to guide the enumeration according to a *schema* given by the user. Intuitively, the schema describes how beats can be cut, *i.e.* what durations are admissible and in which order, thus it is a formal language. Then we run a dynamic-programming algorithm to enumerate lazily all the solutions given by the various divisions of the beat allowed by the schema, ranked according to quality criteria. A user interface allows to prepare the data, select and edit among the results obtained by the algorithm.

Our system is intended to be used in the context of CAC. Thus, it is primarily designed to quantize inputs for which there is no pre-existing score, because the score being composed, as opposed to inputs which are performances of an existing piece. We assume nothing about how the input was generated, our goal is to find the notation that best represents it. In this way, our system is style-agnostic. In particular, it does not use performance models to make up for performance-related imprecisions (such as swing in jazz).

After a brief state of the art, we define in section 3 the schema used, along with the quality criteria and the enumeration algorithm. In section 4, we describe the transcription scenarios made possible by our tools and its user interface. In section 5, we compare our system to existing solutions, and discuss the results.

## 2. STATE OF THE ART

Quantization is an old and complex problem. Many quantization systems exist on the market, integrated in score editors or digital audio workstations (typically, to visualize MIDI data in music sheets). But in most cases, the results are unsatisfactory when the input sequences are too irregular or complex. Besides, the user has very few parameters to influence the result, apart from manually editing it after transcription.

Some systems (in particular the one described in [7]) are based on ratios between successive durations, that must be the ratio of the smallest possible integers. Ali Cemgil et al. proposed a Bayesian model for rhythm transcription [8], in which a performance model with Gaussian noise is used. OpenMusic's current quantization tool, *omquantify* [9], aligns input note onsets on uniform grids in each beat, and chooses the one that gives the best compromise between precision and complexity.

These systems have interesting properties, but they all suggest a unique solution: if the result is unsatisfactory, the algorithm has to be re-run with different parameters.

The OMKant [9] library (not available in recent OpenMusic versions) proposed a semi-supervised approach for segmentation and rhythm transcription. The user could segment the input stream manually or automatically with various algorithms. A variable tempo estimation algorithm placed the beats, and the quantization step was done by *omquantify*. A user interface allowed to set and visualize various parameters (such as the marks used for segmentation), and to choose between the various tempo values suggested by the algorithm.

A framework for score segmentation and analysis in OpenMusic was more recently proposed in [10], which can be used for rhythm transcription as well. This framework allows to segment a note stream to transcribe it with *omquantify* using a different set of parameters on each segment. Such approach provides the user with better control on the final result, and allows more flexibility in specifying the parameters.

## 3. QUANTIZATION ALGORITHM

We first present the problem we want to address and the tools we use for this purpose. Quantization consists in aligning some input points to *grids* of authorized time values. We start by defining formally this problem and then present the formalisms that we shall use for the representation of grids and the output of the problem.

### 3.1 The Quantization Problem

We consider an *input* flow of monophonic (non-overlapping) notes and rests, represented by the increasing sequence of their respective starting dates $\boldsymbol{x} = (x_1, \dots, x_n)$ in an interval $I_0 = [x_0, x'_0[$. Intuitively, for $i \geq 1$, the $i$th event (note or rest) will start at the date $x_i$ and, terminate at $x_{i+1}$ (the starting date of the next event), if $i < n$, or terminate at $x'_0$ if $i = n$. [1]

As an additional input, we also consider a set $G$ of increasing sequences $(z_0, \dots, z_m)$ of dates in the same interval $[x_0, x'_0[$, such that $m \geq 1$, $z_0 = x_0$ and $z_m = x'_0$. Each sequence in $G$ is called a *grid*, and every interval $[z_i, z_{i+1}[$ between two successive points is called a *segment* of the grid. The grid is called *trivial* if $m = 1$. The exact representation of sets of grids is described in Section 3.3.

A quantization *output* is another increasing sequence of dates $\boldsymbol{y} = (y_1, \dots, y_n)$ in the interval $[x_0, x'_0[$, such that

[1] If we want to concatenate $\boldsymbol{x}$ to another input $\boldsymbol{x'}$ in $[x'_0, x''_0[$, then the termination of $x_n$ is set to the first date in $\boldsymbol{x'}$ – these details are left out of this paper.

there exists a grid $(z_0, \ldots, z_m) \in G$ and $y_i$ belongs to $\{z_0, \ldots, z_m\}$ for each $1 \leq i \leq n$. It will be presented as a *rhythm tree*, as explained in Section 3.4.

Our goal is to produce the possible outputs given $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $G$, enumerated according to a fixed weight function which associates a real value to every couple (input ; output) (see Section 3.5).

## 3.2 Uniform Grids

Most quantization algorithms consider a finite set $G$ of *uniform* grids, *i.e.* grids $(z_0, \ldots, z_m)$ whose segments all have the same length: $z_1 - z_0 = z_2 - z_1 = \ldots = z_m - z_{m-1}$. The advantage of this approach is that the number of relevant uniform grids is quite small (typically smaller than 32), hence the number of solutions defined this way is small as well and they can be enumerated in linear time. However, this approach is incomplete and may give unnecessarily complicated results (*e.g.* 7-uplets), in particular when the density of input points is not uniform – see Figure 3.

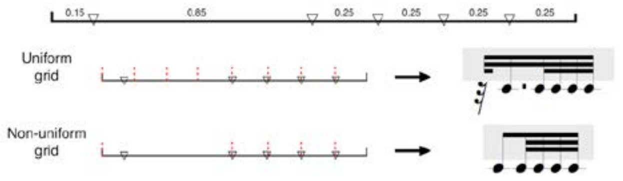


**Figure 3**. Quantization of an input sequence using a uniform grid (division by 8), and a non-uniform grid (division by 2 and then re-division by 4 in the second half only). The non-uniform grid gives a better result because the pitch of the grid is adapted to the density of input points.

## 3.3 Subdivision Schemas and Derivation Trees

For completeness purposes, we use non-uniform grids, defined by recursive subdivision of segments into equal parts. In order to define a finite set of such grids, we use a so-called *subdivision schema*, which is an acyclic context-free grammar $\mathcal{G}$ with a finite set of non-terminal symbols $\mathcal{N}$, an initial non-terminal $N_0 \in \mathcal{N}$ and a unique terminal symbol $\bullet$. The production rules are of two kind : $(i)$ some production rules of the form: $N \to N_1 \ldots N_p$, with $N, N_1, \ldots, N_p \in \mathcal{N}$, and $(ii)$ $\forall N \in \mathcal{N}$, $N \to \bullet$ . The production rules of the latter kind will generally be ommitted.

Defining rhythms with formal grammars and derivation trees [11] (see Figure 4) is quite natural when dealing with common Western music notation, where durations are expressed as recursive divisions of a given time unit.

A derivation with $\mathcal{G}$ consist in the successive replacement of non-terminals $N$ by the corresponding right-hand-side of production rules, starting with $N_0$. Intuitively, during a replacement, the non terminal $N$ correspond to a segment of the grid (an interval), and either $(i)$ the application of $N \to N_1 \ldots N_p$ is a division of this segment into $p$ equal parts, or $(ii)$ the application of $N \to \bullet$ corresponds to not dividing any further.

Every such derivation is represented by a *derivation tree* (DT) whose leaves are labeled with $\bullet$ and inner nodes are labeled with non-terminals of $\mathcal{N}$. The labels respect the

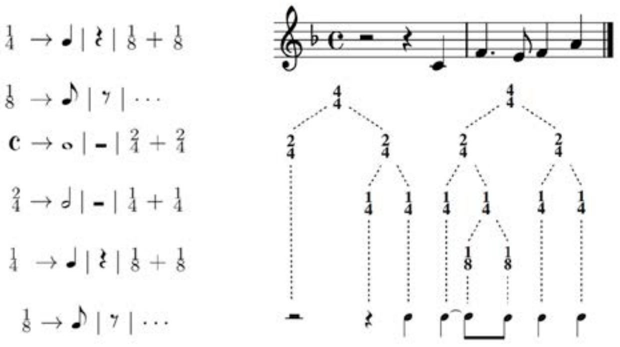


**Figure 4**. [11] A context-free grammar, a rhythm and the corresponding derivation tree.

production rules in a sense that if an inner node $\nu$ is labelled with $N$ and its sons $\nu_1, \ldots, \nu_p$ are respectively labelled with $N_1, \ldots, N_p$, then there exists a production rule $N \to N_1 \ldots N_p$.

Given a DT $t$ of $\mathcal{G}$ and an initial interval $I_0 = [x_0, x'_0[$, we associate an interval to each node of $t$ as follows:

to the root node $\nu_0$, we associate $I_0$,

if $\nu$ is an inner node associated to $I = [z, z'[$ and with $p$ sons $\nu_1, \ldots, \nu_p$, then $\nu_i$ is associated with :
$$part(I, i, p) := \Big[z + \tfrac{(i-1)(z'-z)}{p}, z + \tfrac{i(z'-z)}{p}\Big[.$$

The grid $g_t$ associated to a DT $t$ of $\mathcal{G}$ is defined by the bounds of its segments, which are the intervals associated to the leaves of $t$. Following the presentation in Section 3.1, we make by abuse no distinction between a schema $\mathcal{G}$, its set of DTs and the set of associated grids $G$.

The quantization output $\boldsymbol{y} = (y_1, \ldots, y_n)$ associated to an input $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a DT $t$ is defined as the $n$ closest point to $x_1, \ldots, x_n$ in the grid $g_t$ associated to $t$ (with a default alignment to the left in case of equidistance). Given an input $\boldsymbol{x}$ and a schema $\mathcal{G}$, the set of *quantization solutions* is the set of output $\boldsymbol{y}$ associated to $\boldsymbol{x}$ and a DT of $\mathcal{G}$.

## 3.4 Solutions as Sets of Rhythm Trees

Searching the best alignments of the input points to some allowed grids is a classical approach to rhythm quantization. However, instead of computing all the possible grids $g_t$ represented by $\mathcal{G}$ (as sequences of points) and the alignments of the input $\boldsymbol{x}$ to these grids, we compute only the DTs, using dynamic programming techniques.

Indeed, trees, rather than sequences, are the structures of choice for the representation of rhythms in state-of-the-art CAC environments such as OpenMusic [12] .

Our DTs (*i.e.* grids) are converted into OpenMusic rhythm trees (RT) for rendering purpose and further use by composers. The conversion is straighforward, using decoration of the leaves of DTs and tree transformation functions.

## 3.5 Rhythm Tree Series

In order to sort the solution set, we consider the notion of *trees series* [13], which is a function associating to each tree a weight value in a given domain—here the real numbers. In our case, the smaller the weight of a tree is, the better the corresponding notation is. We describe below the definition of this function as a combination of several criteria.

### *3.5.1 Criteria and Combinations*

The weight of a tree is calculated by combining several criteria, which are functions associating a real value to an input $\boldsymbol{x}$ and a DT $t$. We take into account a distance criterion, and a complexity criterion. They are computed recursively: the value of a criterion for a tree is evaluated using the values of criteria of his son.

The chosen *distance* criteria is defined by :

$$\mathit{dist}(\boldsymbol{x}, t) = \sum_{x_i \in \mathit{segment}(t)} |x_i - y_i|$$

for a subtree $t$ which is a leaf, where $\mathit{segment}(t)$ is the set of inputs $x_i$ contained in the interval associated to $t$, and $\boldsymbol{y}$ is defined as above, and

$$\mathit{dist}\big(a(t_1, \dots, t_p)\big) = \sum_{i=1}^{p} \mathit{dist}(t_i)$$

The *complexity* criterion is defined as a combination of several sub-criteria. The first sub-criterion is related to the size of the tree and the degrees of the different nodes. It is the sum of the numbers of nodes having a certain degree, weighted by penalty coefficients. We denote by $\beta_j$ the coefficient describing the complexity of the degree $j$, and follow the order recommended in [14] to classify arities from the simplest to the most complex: $\beta_1 < \beta_2 < \beta_4 < \beta_3 < \beta_6 < \beta_8 < \beta_5 < \beta_7$... The other sub-criterion is the number of *grace notes* present in the final notation. A grace note corresponds to the case where several entry points are aligned on the same grid point, *i.e.* $y_{i+1} = y_i$ (we recall that we consider monophonic inputs, two notes aligned to the same point do not correspond to a chord). We aim at minimizing the number of grace notes, since too many grace notes hinder readability.

If $t$ is a leaf, then $\mathit{comp}(t) = g(t)$, the number of grace notes, determined by counting the number of points of the segment aligned with each of its boundaries, and

$$\mathit{comp}\big(a(t_1, \dots, t_p)\big) = \beta_p + \sum_{i=1}^{p} \mathit{comp}(t_i)$$

The weight $w(t)$ of a tree $t$ is a linear combination of the above criteria:

$$w(t) = \alpha.\mathit{dist}(t) + (1 - \alpha).\mathit{comp}(t)$$

The coefficient $\alpha$ is a parameter of the algorithm, used to adjust the relative importance of the two criteria in the final result: $\alpha = 0$ will return results favoring simplicity of notation (small rhythm trees, simple tuplets, few grace notes) at the expense of the fitness of transcription, while $\alpha = 1$ will return rhythm trees of maximum depth, often corresponding to less readable notations, but transcribing the input as accurately as possible.

### *3.5.2 Monotonicity*

The criteria and their combination were not chosen arbitrarily, they were chosen to follow the following property of monotony for the purpose of correctness of the enumeration algorithm below.

$$\forall t = a(t_1, \dots, t_p)\, \forall i \in [1..p]\, \forall t'_i\, w(t'_i) > w(t_i) \Rightarrow \\ w(a(t_1, \dots, t_{i-1}, t'_i, t_{i-1}, \dots, t_p) > w(a(t_1, \dots, t_p)$$

In other words, if we replace a subtree by another sub-tree of greater weight, the weight of the super-tree will also be greater. One can check that this property holds for the functions defined as above.

## 3.6 Enumeration Algorithm

A given subdivision schema $\mathcal{G}$ will generaly define an exponential number of non-uniform grids, and therefore an exponential number of quantization solutions, according to the definition in Section 3.4. Hence, we would like to avoid having to compute all of them before ranking them, we want to lazily enumerate them in increasing weight. More precisely, the following dynamic programming algorithm called $k$-*best* [15] returns the solutions by packets of $k$, where $k$ is a constant fixed by the user.

### *3.6.1 Enumeration Table*

The $k$-best algorithm is based on a table $T$ built from the schema $\mathcal{G}$. Each key of this table has the form $\langle N, I\rangle$, where $N$ is a non-terminal of $\mathcal{G}$ and $I$ is an interval $[z, z'[$ associated with a node labeled with $N$ in an DT of $\mathcal{G}$. Every entry $T[N, I]$ of the table contains two lists:

***best*-list** $\mathit{bests}[N, I]$, containing the minimal weighted sub-DT whose root is labeled with $N$ and associated the interval $I$, together with their weight and the values of $\mathit{dist}$ and $\mathit{comp}$.

***candidate*-list** $\mathit{cands}[N, I]$, containing sub-DT, among which the next best will be chosen (some weights might not have been evaluated yet).

The sub-DT in the above lists are not stored in-extenso but each of them is represented by a list of the form $(\langle N_1, i_1\rangle, \dots, \langle N_p, i_p\rangle)$, called a *run*. A run in one of the two lists of $T[N, I]$ represents a sub-DT whose root is labeled with $N$ and with $p$ children, such that the $j$th child is the element number $i_j$ in the best-list of $T[N_j, \mathit{part}(I, j, p)]$. The pair $\langle N_j, i_j\rangle$ is called a link to the $i_j$-best of $\langle N_j, \mathit{part}(I, j, p)\rangle$.

### *3.6.2 Initialization of the Table*

The initialization and update of the list of candidates exploits the hypothesis of monotony of the weight function mentioned in Section 3.5.2. Indeed, this property implies that the best tree (tree of lesser weight) will be built with the best children, and therefore will be represented by a run containing only links to 1-bests. More precisely, we initialize every entry $T[N, I]$ with an empty best-list and a candidate-list containing one run $\big(\langle N_1, 1\rangle, \dots, \langle N_p, 1\rangle\big)$ for each production rule $N \to N_1 \dots N_p$ of $\mathcal{G}$ (division in $p$ parts), and one empty run (), which corresponds to the case of a leaf in the DT (end of divisions). The weights of the runs in the initial candidate-lists is set as unknown.

As an optimization, when the intersection of the input $\boldsymbol{x}$ with $I$ is empty, then we only put () in the candidate list. Indeed, there is no need to further divide a segment containing no input points.

### *3.6.3 Algorithm*

The enumeration algorithm is described in details in [16]. It reuses the results already computed and builds the trees in a lazy way: as indicated above, thanks to the monotony of the weight function, in order to construct the best tree, one just needs to construct the best sub-trees. The algorithm works recursively: in order to evaluate the weight of a sub-DT, it will evaluate the weight of each of its children. The main function $best(k, N, I)$ is recursive. It returns the $k$-best DT of $\langle N, I\rangle$, given $k$, $N$ and $I$:

**1.** If the best-list of $T[N, I]$ contains $k$ elements or more, then return the the $k$th run of this list, together with its associated weight and criteria values.

**2.** Otherwise, evaluate the weight of all the candidates in $T[N, I]$ as follows: for a run $\big(\langle N_1, i_1\rangle, \ldots, \langle N_p, i_p\rangle\big)$ in the candidate-list of $T[N, I]$ whose weight is unknown, call recursively $best\big(i_j, N_j, part(I, j, p)\big)$ for each $1 \leq j \leq p$, and then evaluate the weight and criteria values, using the values returned for its children and the equations in Section 3.5.1.

**3.** Once all the weights of the runs in the candidate-list of $T[N, I]$ have been evaluated, remove the run $\big(\langle N_1, i_1\rangle, \ldots, \langle N_p, i_p\rangle\big)$ of smallest weight from this list, add it to the best-list of $T[N, I]$ (together with weight and criteria values), and then add to the candidate-list the following next runs, with unknown weight:
$\big(\langle N_1, i_1 + 1\rangle, \langle N_2, i_2\rangle, \ldots, \langle N_p, i_p\rangle\big)$,
$\big(\langle N_1, i_1\rangle, \langle N_2, i_2 + 1\rangle, \ldots, \langle N_p, i_p\rangle\big), \ldots,$
$\big(\langle N_1, i_1\rangle, \ldots, \langle N_{p-1}, i_{p-1}\rangle, \langle N_p, i_p + 1\rangle\big)$.

An invariant of the algorithm is that for each entry of $T$, the next best tree (after the last best tree already in the best-list) is represented in the candidate list. This property stems from the monotonicity of the weight function, and ensures the completeness of the algorithm.

The algorithm is exponential in the depth of the schema (the maximal depth of a derivation tree). This value is typically 4 or 5 for obtaining usual rhythm notation. On our experiments with inputs of 10 to 20 points, the table has typically a few hundred entries (depending of the size of the schema).

### 3.7 Coupled Quantization and Tempo Estimation

Calling $best(k, N_0, I_0)$ will return the $k^{\text{th}}$ best quantization of an input $\boldsymbol{x}$ in the interval $I_0$, according to the given schema $\mathcal{G}$. It works when the tempo is known in $I_0$. Estimating the tempo is a difficult problem; we propose a simple solution that gives good results in our context, by coupling tempo estimation with the quantization algorithm.

The idea is, given $\boldsymbol{x}$, $I_0 = [x_0, x'_0[$ and $\mathcal{G}$, to add to $\mathcal{G}$ a preliminary phase of division of $I_0$ into a certain number $m$ of equal parts, corresponding to beats. This tempo estimation makes the hypothesis that the tempo is constant over $I_0$. The values for $m$ are chosen so that the corresponding tempo is between values $\theta_{min}$ and $\theta_{max}$, typically 40 and 200 bpm, or any range specified by the user:

$$\frac{(x'_0 - x_0) \times \theta_{min}}{60} \leq m \leq \frac{(x'_0 - x_0) \times \theta_{max}}{60} \qquad (1)$$

where $x_0$ and $x'_0$ are assumed given in physical time (seconds). This is done by constructing $\mathcal{G}'$, by addition to $\mathcal{G}$ of a new initial non-terminal $N'_0$ and new production rules $N'_0 \to \underbrace{N_0, \ldots, N_0}_{m}$ for all integral values of $m$ satisfying (1). Using this new schema $\mathcal{G}'$, and adapted weight functions and criteria, the above enumeration algorithm will return the $k$ best transcription solutions for all tempi, obtained by a coupled estimation of tempo and quantization. A variant consists in enumerating the $k$ best quantization solutions for each tempo given by the possible values of $m$.

## 4. INTEGRATION IN OPENMUSIC

The algorithm presented in the previous section has been implemented as an OpenMusic library. A user interface (Figure 5) has also been developed to monitor and make use of the results of this algorithm.

The user first loads as input stream $\boldsymbol{x}$ a CHORD-SEQ object, which is the analogous of a MIDI file in terms of time representation (the start and end dates of notes are expressed in milliseconds). The rhythmic transcription is then performed in 4 steps.

**1. Segmentation:** The user segments the input stream of notes $\boldsymbol{x}$ in the top left panel. Typically, the length of a segment should be in the order of one bar. Moreover, these segments should preferably correspond to regions where the tempo is constant (see Section 3.7).[2] The user can also specify different parameters for each segment, such as the subdivision schemas and tempi bounds.

**2. Quantization:** The algorithm is run on each segment independently, to compute the $k$ best solutions.

**3. Choice of a solution:** The user sees the $k$ best transcriptions in the right panel, and selects, for each segment, one of them. The $dist$ values for each transcription are indicated. The selected transcriptions are then concatenated and displayed in the bottom left panel.

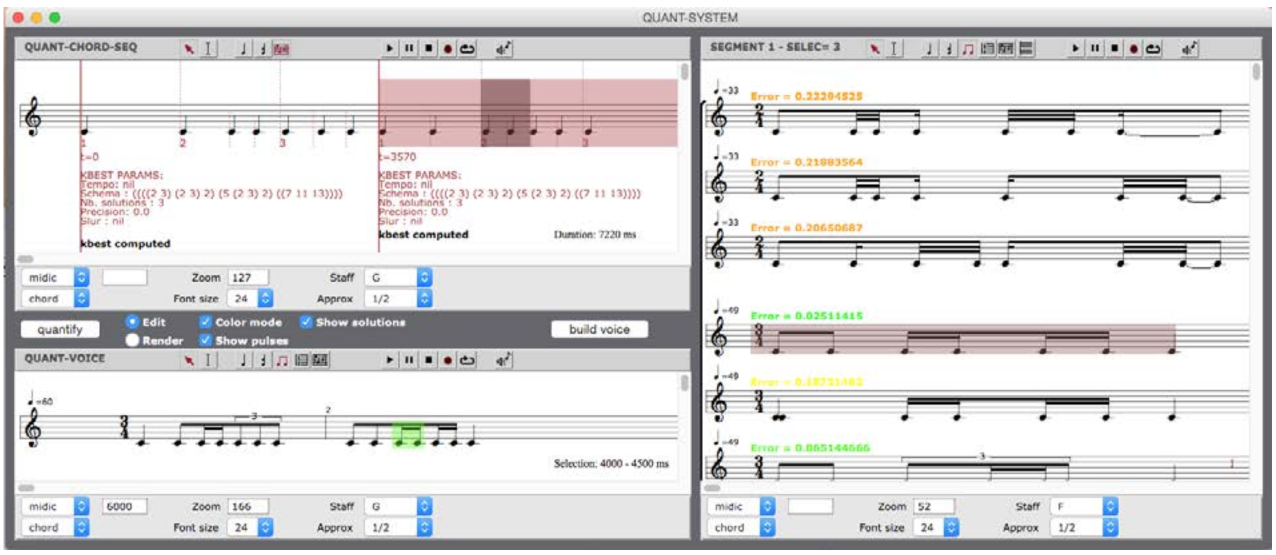
**4. Edition of the solution:** The user can edit the chosen solution, with the content of the table $T$ used by the quantization algorithm. When he/she selects one region corresponding to a sub-tree in the transcription, he can visualize the *best*-list for this region and choose in the list an alternate solution for it to be replaced in the final score. The user can also estimate the $dist$ value for each sub-tree via a color code. At any time he/she can request to extend the list with the following $k$ best solutions for this sub-tree.[3]

## 5. EVALUATION

Automated evaluation of the system we presented is difficult, as there is no unique objective criteria to evaluate a "good" transcription. One method could be to quantize performances of given pieces and check if the system outputs the original score. This method has two disadvantages. First, our approach is interactive: the right result will not necessarily be the first one, but it might be among the transcriptions given by the algorithm. If not, we could easily obtain it by editing the solution. Rather than counting the number of matching first result, we should count the number of operations to obtain the targeted score. Moreover, our system was not designed for performance transcription. The constant-tempo hypothesis is not adapted in this case: a variable-tempo model would yield better

[2] An automatic constant-tempo regions segmentation algorithm is currently under development.

[3] A video demonstration of the system is available at `http://repmus.ircam.fr/cao/rhythm/`.



t=0
KBEST PARAMS:
Tempo: nil
Schema : ((((2 3) (2 3) 2) (5 (2 3) 2) ((7 11 13))))
Nb. solutions : 3
Precision: 0.0
Slur : nil
kbest computed

t=3570
KBEST PARAMS:
Tempo: nil
Schema : ((((2 3) (2 3) 2) (5 (2 3) 2) ((7 11 13))))
Nb. solutions : 3
Precision: 0.0
Slur : nil
kbest computed

Duration: 7220 ms

midic, chord, Zoom 127, Font size 24, Staff G, Approx 1/2

Selection: 4000 - 4500 ms

midic 6000, chord, Zoom 166, Font size 24, Staff G, Approx 1/2

Error = 0.22204525
Error = 0.21883564
Error = 0.20650687
Error = 0.02511415
Error = 0.18721482
Error = 0.065144666

midic, chord, Zoom 52, Font size 24, Staff F, Approx 1/2

**Figure 5**. Screenshot of the transcription system's user interface in OpenMusic

results and thus, our system's results will not be representative of its quality. We are currently favoring test sessions with composers, in order to assess the relevance and user-friendliness of our system.

For illustration purposes, let us consider the input series of durations is the rhythm given in Figure 1c), to which we added a uniform noise between -75 and 75ms. We transcribed it with *omquantify*, with the score editor Sibelius 6 via the MIDI import function, and with our system. The results are shown in Figure 6.

Sibelius outputs a result that is easily readable, but quite far from the input score. There is no tempo estimation, the tempo of the MIDI file is directly used to quantize. However, in many cases, when tempo is not known, the MIDI file's tempo is set to default (here, 60 beats per minute), which leads to an incorrect transcription. Moreover, many durations are badly approximated: the third and fourth notes are supposed to be of same duration, as well as the 4 sixteenth notes at the end.

*omquantify* outputs good results, but to obtain them, the user has to input the right tempo and the right signature (3/4) as parameters. There is no tempo estimation in *omquantify*, and finding the right tempo can be very tricky, as discussed in Section 1. Otherwise, with default parameters (tempo = 60, signature = 4/4), the results might be exact, but it is inexploitable, as it is too complicated.

With default parameters ($\alpha = 0.5$), and with an adequate segmentation, our system gives good results. The estimated tempo is very close to the original (though not exact in the second bar), and the right signature is found automatically. The note values are also very close to the original, except for the triplet. Nevertheless, it should be underlined that the solution shown here is only the first proposition made by the algorithm. The goal transcription can in fact be obtained by choosing the third proposition for the first bar, and the first for the second bar. Besides, by changing the $\alpha$ parameter, we can get the goal rhythm as the first proposition made by the algorithm.

These good results rely on a good preliminary segmentation : we cut the input stream according to the original

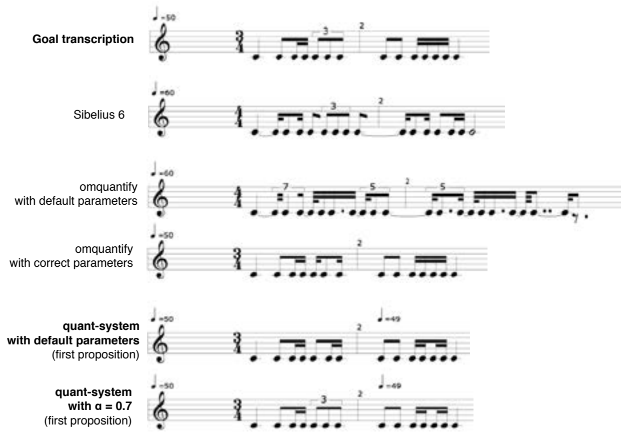


**Figure 6**. Results of quantization with various systems. Sibelius gives a result quite far from the input. *omquantify* gives good results, but the user has to find the right set of parameters. Our system works well with default parameters (the first proposition is not exact, the third is), and can give even better results by adjusting them.

bars. As our system considers each segment as a bar, we wouldn't have had the same result with a different segmentation. Moreover, if the segmentation marks hadn't been placed on a note corresponding to a beat of the original input, the tempo estimation step might not have worked as well (hence the importance of the supervised aspect of the system). Besides, the enumeration algorithm, and thus the tempo estimation, is ran on each segment independently, which is why we have a small tempo difference between the two bars.

The results are also less satisfactory when segments are too long (more than a few bars). Indeed, a long segment entails a lot of possible tempo values (cf section 3.7), and thus, computations are longer, and it is more difficult to choose the right tempo value.

In general, the following interactive workflow has shown to be quite successful: the user ranks the solutions by complexity (with a small parameter $\alpha$, see Section 3.5.1), and then refines the more difficult (dense) regions, choosing more complex and accurate alternative solutions for these regions. The advantage of this method is that since complexity and precision are antagonist criteria, the results will be ranked by complexity and also approximately by precision, and thus there is a relative regularity in the ordering of the solutions, which makes exploration easier. On the contrary, when $\alpha$=0.5, some very precise and complex solutions can be ranked close to imprecise and simple solutions, as they may have similar weights.

## 6. CONCLUSION

We have presented a new system for rhythmic transcription. The system ranks the transcription solutions according to their distance to input and complexity, and enumerates them with a lazy algorithm. An interface allows the user to choose from the available transcriptions and edit it in a semi-supervised workflow. At the time of this writing this tool is being tested with composers.

Finding relevant parameters ($\alpha$, and $\beta_j$'s) is a sensitive problem in our approach. One could use for this purpose a corpus of pairs performance/scores, such as *e.g.* the Kostka-Payne corpus [17], in order to learn parameter values that maximize fitness to some extend (number of correct transcriptions in first rank, number of correct transcriptions in the first $n$ solutions...). We could get around the problem of variable-tempo performances by segmenting the input stream in beats, in order to focus on quantization only.

The choices made by the user could also be used to learn some user preferences and improve the results of transcription, as it was proposed in [18]. For example, the user could specify the solution he wants to keep, and the solutions he doesn't want to see again. This information can then be used to adapt the $comp$ function so that the kept solutions have a smaller weight, and the unwanted ones have a higher weight.

Finally, an alternative approach consist in considering a vectorial weight domain, partially ordered by component-wise comparison, and enumerate the Pareto front (*aka skyline* [19]).

**Acknowledgments**

This project was supported by the project with reference ANR-13-JS02-0004 and IRCAM's "rhythm" UPI.

We wish to thank the composers who help us in the design and evaluation of our tool, in particular Julia Blondeau, Karim Haddad, Daniele Ghisi and Mikhail Malt.

## 7. REFERENCES

[1] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer-assisted composition at IRCAM: From PatchWork to OpenMusic," *Computer Music Journal*, vol. 23, no. 3, pp. 59–72, 1999.

[2] M. Piszczalski and B. A. Galler, "Automatic music transcription," *Computer Music Journal*, vol. 1, no. 4, pp. 24–31, 1977.

[3] M. A. Alonso, G. Richard, and B. David, "Tempo and Beat Estimation of Musical Signals," in *Proc. Int. Society for Music Information Retrieval Conf. (ISMIR)*, Barcelona, Spain, 2004.

[4] A. Klapuri *et al.*, "Musical meter estimation and music transcription," in *Cambridge Music Processing Colloquium*, 2003, pp. 40–45.

[5] A. T. Cemgil, "Bayesian Music Transcription," Ph.D. dissertation, Radboud Universiteit Nijmegen, 2004.

[6] P. Desain and H. Honing, "The quantization of musical time: A connectionist approach," *Computer Music Journal*, vol. 13, no. 3, pp. 56–66, 1989.

[7] D. Murphy, "Quantization revisited: a mathematical and computational model," *Journal of Mathematics and Music*, vol. 5, no. 1, pp. 21–34, 2011.

[8] A. T. Cemgil, P. Desain, and B. Kappen, "Rhythm quantization for transcription," *Computer Music Journal*, vol. 24, no. 2, pp. 60–76, 2000.

[9] B. Meudic, "Détermination automatique de la pulsation, de la métrique et des motifs musicaux dans des interprétations à tempo variable d'oeuvres polyphoniques," Ph.D. dissertation, UPMC - Paris 6, 2004.

[10] J. Bresson and C. Pérez Sancho, "New framework for score segmentation and analysis in OpenMusic," in *Proc. of the Sound and Music Computing Conf.*, Copenhagen, Denmark, 2012.

[11] C. S. Lee, "The Rhythmic Interpretation of Simple Musical Sequences: Towards a Perceptual Model," *Musical Structure and Cognition*, vol. 3, pp. 53–69, 1985.

[12] C. Agon, K. Haddad, and G. Assayag, "Representation and Rendering of Rhythm Structures," in *Proc. 2nd Int. Conf. on Web Delivering of Music*. Darmstadt, Germany: IEEE Computer Society, 2002, pp. 109–113.

[13] Z. Fülöp and H. Vogler, "Weighted Tree Automata and Tree Transducers," in *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Volger, Eds. Springer, 2009, pp. 313–403.

[14] C. Agon, G. Assayag, J. Fineberg, and C. Rueda, "Kant: A critique of pure quantification," in *Proc. of ICMC*, Aarhus, Denmark, 1994, pp. 52–9.

[15] L. Huang and D. Chiang, "Better k-best parsing," in *Proc. of the 9th Int. Workshop on Parsing Technology*. Association for Computational Linguistics, 2005, pp. 53–64.

[16] A. Ycart, "Quantification rythmique dans OpenMusic," Master's thesis, UPMC - Paris 6, 2015.

[17] D. Temperley, "An evaluation system for metrical models," *Computer Music Journal*, vol. 28, no. 3, pp. 28–44, 2004.

[18] A. Maire, "Quantification musicale avec apprentissage sur des exemples," IRCAM, Tech. Rep., 2013.

[19] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. of the 17th Int. Conf on Data Engineering*. Heidelberg, Germany: IEEE, 2001, pp. 421–430.

```
/* Returns the k-th best run for entry T[N,I], along with its weight                  */
Function best ($k, N, I$):
    if $length(bests[N,I]) \geq k$ then
        return $bests[N,I][k]$, $w(bests[N,I][k])$;
    else
        if $cands[N,I]$ is empty then
            /* the k-best cannot be constructed, no more candidates                  */
            return error
        else
            run := min($cands[N,I]$) ;
            if w(run) is known then
                add run to $bests[N,I]$;
                remove run from $cands[N,I]$;
                add next candidates to $cands[N,I]$ with unknown weights;
                /* cf section 3.6.3                                                   */
                best($k, N, I$);
                /* iterate until $k$ bests are constructed                            */
            else
                eval(run,$N, I$);
                best($k, N, I$);
                /* iterate until all weights are known                                */
            end
        end
    end

/* Evaluates the weight of a given run, and updates it in the cands list              */
Function eval (run,$N, I$):
    if run = () then
        /* the node is a leaf, no further subdivision                                 */
        compute w(run);
        /* cf section 3.5.1                                                           */
        update w(run) in $cands[N,I]$;
    else
        /* the node has sons,recursive call to best                                   */
        let run = $\big(\langle N_1, i_1\rangle, \ldots, \langle N_p, i_p\rangle\big)$;
        weights := (best$\big(i_1, N_1, part(I,1,p)\big), \ldots$ ,best$\big(i_p, N_p, part(I,p,p)\big)$);
        w(run) = sum of weights;
        /* cf section 3.5.1                                                           */
        update w(run) in $cands[N,I]$;
    end
```

**Algorithm 1:** Pseudo-code of the enumeration algorithm