Rhythm Inference Helping Writing Music Scores

François Schwarzentruber
University of Rennes, IRISA, CNRS, France
francois.schwarzentruber@ens-rennes.fr

Abstract

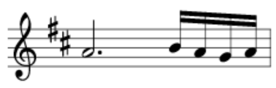
We present a new problem called rhythm inference. It consists in inferring the duration of each note and each rest from a partial specification. We then formulate rhythm inference as a constraint satisfaction problem and we use mixed linear programming to solve it. The solution is implemented for a language representing music scores, called `abcd`. Interestingly, the rhythm is inferred in real-time from partial musical indications.

1 Introduction

Editing music scores on a computer can be done essentially by two means: graphical editors (e.g. Musescore) or textual forms (e.g. Lilypond, ABC). However, it is often tedious to edit rhythm, i.e. to specify/modify the duration of each note and each rest in a music score[1]. Furthermore, the representation in Lilypond or ABC of a score is verbose and requires some expertise.

In this paper, we propose a solution where the user only *partially specify* the rhythm. As it is complicated to represent a partial specification graphically, the input is a text. Our algorithmic solution is implemented in a language, called `abcd`[2]. Moreover, the language `abcd` has the same spirit as the *markdown* language used to represent documents. Indeed, its syntax is concise and looks like a real music score.

Example 1. *In order to write the following music score:*



the user writes the following `abcd` *code:*

```
𝄞  ## 4/4  a.        b a g a |
```

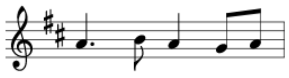
As you see in the example, in `abcd`, you may only specify partially the durations of the notes; and the textual form is suitable for that purpose. More precisely, the durations are inferred from the amount of space left between notes, and also from musical indications. The dot (`.`) on the first note `a.` indicates that its duration is a standard duration (1, 1/2, 1/4, 1/8, etc.) augmented with its half (i.e. we get 3/2, 3/4, 3/8, etc.). The last four notes seem to have smaller and equal durations. The system then infers some rhythm that is consistent with the `abcd` textual input.

If the user changes the amount of space between notes then the system outputs another score.

Example 2. *The following code is obtained from the one in Example 1 but by reducing the length of the first* a *and increasing the length of the second* a*:*

```
𝄞  ##  4/4  a. b a  g a |
```

It leads to the following different score:



Our contribution is the definition of the so-called *rhythm inference problem*, and its implementation. Rhythm inference is the mechanism of guessing the durations of the notes and the rests from the spacing and some partial rhythmic indications. Using rhythm inference has several advantages:

1. The source code in `abcd` is more succinct than in Lilypond or in ABC. Thus, it is more readable.
2. It avoids boredom when writing music since the user does not have to specify the duration of each note anymore.
3. As shown in Example 2, it is convenient to modify an existing rhythm. The user just has to add/remove some indications in the source code; e.g. add/remove dots (`.`), add/remove/change measure bars (|), etc. This advantage is especially interesting in combination with automatic transcription, which consists in automatically computing the rhythm from a human performance. As the output of automatic transcription often contains errors, it is relevant to have a language with rhythm inference in order to correct the errors more easily.
4. In the output, each measure is guaranteed to last the total duration specified by the signature (e.g. 4/4) and the spacing between notes.

[1]or in Lilypond for a sequence of notes that will then have all the same durations

[2]See https://github.com/francoisschwarzentruber/abcd for the specification of the language `abcd`.

Outline. In Section 2, we define formally the problem of *rhythm inference*. In Section 3, we present how the tool is implemented. In Section 4, we present the demonstration. In Section 5 we discuss the related work. In Section 6 we discuss the future work.

2 Rhythm Inference

Let us start with an informal description and some examples. We then explain how the problem is formalized as a constraint satisfaction problem. We then explain how it is transformed into a Mixed Linear Program.

2.1 Informal Description

Rhythm inference is by its nature a constraint satisfaction problem. Its problem is first made of approximative durations $\hat{\delta} = (\hat{\delta}_1, \dots, \hat{\delta}_n) \in {\mathbb{R}^+}^n$, where $\mathbb{R}^+$ is the set of the strictly positive real numbers. In our case, $\hat{\delta}_i$ is just the number of spaces between the i-th note and the $i+1$-th note (or between the n-th note and the measure bar | if $i = n$). In Example 1, we have $\hat{\delta} = (9, 1, 1, 1, 1)$.

The second part of the input consists of the finite domains of the possible durations of the notes. Consider again Example 1. The last four notes have no special rhythm indications. So their corresponding domains are the set of standard note durations. According to the following table of given the durations of standard notes

Notes:	𝅝	𝅗𝅥	♩	♪	...
Durations:	1	1/2	1/4	1/8	...

we set

$$\Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = \{1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64\}.$$

The first note `a.` is however dotted. So the corresponding domain is obtained from the standard domain by multipling by $3/2$ all the possible durations:

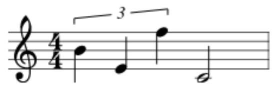
$$\Delta_1 = \{3/2, 3/4, 3/8, 3/16, 3/12, 3/64\}.$$

If a note is specified to be part of a tuplet, the set Δ_i is adapted accordingly: we multiply its original durations by $1/k$ for a k-tuplet. The following example illustrates how the sets Δ_i are changed for a triplet.

Example 3. *With the following code*

```
𝄞   4/4   (3  b  e  f′  c        |
```

we set $\Delta_1 = \Delta_2 = \Delta_3 = \{1/3, 1/6, 1/12, \dots\}$. *Indeed, we multiplied the original durations by 1/3. This code produces*



In `abcd`, the duration of a note can also be completely specified in the user wants to. For instance if the i-th note is `a2`, the `2` means that the note is a half-note 𝅗𝅥, thus the corresponding domain is a singleton: $\Delta_i = \{1/2\}$.

To sum up, the textual input is transformed into an instance $(\hat{\delta}, \Delta_1, \dots, \Delta_n)$. Now the goal is to infer some *consistent* durations $\delta = (\delta_1, \dots, \delta_n) \in {\mathbb{R}^+}^n$ with respect to $\hat{\delta}$ and $\Delta_1, \dots, \Delta_n$. What is still pending is a formal definition of consistency.

2.2 Formal Description

We propose to start our model of the notion of consistent durations as a mathematical program with these constraints:

$$\delta_i \in \Delta_i \text{ for all } i = 1..n \quad (1)$$

$$\sum_{i=1}^{n} \delta_i = T \quad (2)$$

Constraint (1) says that each variable belongs to its domain. Constraint (2) says that the sum of the duration equals the duration T of a measure. For a 4/4 measure, $T = 1$. For a 3/4 measure, $T = 0.75$.

Now, we have to take care of the number of spaces $\hat{\delta}_1, \dots, \hat{\delta}_n$. The first naïve idea would be to minimize the closeness between the variable $\delta_1, \dots, \delta_n$ and the approximative durations $T\hat{\delta}_1/s, \dots T\hat{\delta}_n/s$ where s is the total number of spaces; for instance minimize $\sum_{i=1}^{n} |T\hat{\delta}_i/s - \delta_i|$. However, the issue is that $T\hat{\delta}_i/s$ does not represent properly the approximation duration of the i-th note. The scale of the spaces is approximate and that is precisely the feature that makes our approach robust.

Instead we focus on relative comparisons of durations. Suppose $\hat{\delta}_i \geq \hat{\delta}_j$. Then ideally we should have $\delta_i \geq \delta_j$; meaning that if $\delta_i < \delta_j$ then $\delta_j - \delta_i$ should be as small as possible. To model this constraint, we introduce an error variable err_{ij}. The obtain mathematical program (*) is:

$$\text{minimize } \sum_{i,j=1}^{n} err_{ij}$$
$$\begin{cases} (1)\ (2) \\ \delta_j - \delta_i \leq err_{ij} \text{ if } \hat{\delta}_i \geq \hat{\delta}_j \end{cases}$$

The variables of the program are: for all $i \in \{1, \dots, n\}$, a finite-domain variable δ_i, and for all $i, j \in \{1, \dots, n\}$ with $\hat{\delta}_i \geq \hat{\delta}_j$, a positive real number variable err_{ij}.

Definition 1. *Rhythm inference is the algorithmic problem defined as follows:*

- *input: approximative durations* $\hat{\delta}_1, \dots, \hat{\delta}_n \in \mathbb{R}^+$, *finite domains* $\Delta_1, \dots, \Delta_n \subseteq \mathbb{R}$, *duration* T *of a measure;*
- *output: inferred durations* $\delta_1, \dots, \delta_n$ *that form a solution of the mathematical program (*).*

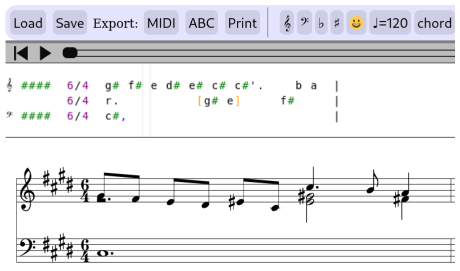
2.3 Reduction to Mixed Linear Programming

The mathematical program (*) described above can be transformed into a mixed linear program as follows. For all $i = 1..n$, for all possible duration $d \in \Delta_i$, we introduce a Boolean variable $x_{id} \in \{0, 1\}$. The intuitive meaning of $x_{id} = 1$ is $\delta_i = d$. We add the constraint $\sum_{d \in \Delta_i} x_{id} = 1$ saying that δ_i takes a unique value. We then replace each occurrence of δ_i in constraints by $\sum_{d \in \Delta_i} d \times x_{id}$.

The obtained linear program is mixed: the variables err_{ij} are positive real number variables, while the x_{ij} are Boolean variables.

3 Tool Description

The graphical user interface is divided in two parts (see Figure 1): the top is the text editor for the code written



```
𝄞 ####   6/4   g# f# e d# e# c# c#'.     b a   |
         6/4   r.         [g# e]      f#        |
𝄢 ####   6/4   c#,                              |
```

Figure 1: Screenshot of the tool. On the top: the `abcd` code. On the bottom: the output.

in `abcd`; the bottom is the corresponding generated music score. The graphical user interface is written in standard HTML/JavaScript/CSS. The display of the music score is performed by the library abcjs[3].

Roughly speaking, the tool takes the score description written in our markdown language `abcd`. For each measure, the system produces an instance of the rhythm inference problem. Some code in Python transforms it into the mixed linear program (*), which is then solved with Google OR tools[4]. The obtained optimal solution of (*) is transformed into some ABC code in which all rhythms are explicitly written. The ABC code is then rendered into a music score image which is the final output.

4 Interaction with the Tool

The user starts a score by choosing the instrument setting (piano solo, flute solo, choir, etc.). Then user can choose to enter notes either by typing text from the computer keyboard, or playing on a MIDI musical keyboard,

The user can edit the score in order to change the rhythm: adding/removing some amount of space between notes, adding/removing rhythm indication (dots, nuplets), change the time signature (e.g. 4/4 to 3/4). The user sees the output music score in *real-time* while making his/her modifications.

The tool is available on Github [5]. The interested reader may also watch videos about the tool, available from the Github project.

5 Related Work

Rhythm inference has some link with *automatic transcription* of music whose aim is to produce a music score from a human performance. In the two problems – rhythm inference and automatic transcription – the goal is to obtain a rhythm from a somehow fuzzy input.

Music score editors often implement so-called *quantization* for solving automatic transcription because it is simple to implement: it consists in aligning musical notes/rests to a predefined grid, typically based on a rhythmic subdivision. There are two drawbacks: the human is forced to play with a regular and rigid tempo in order to hope obtaining a suitable music score; and often the resulting music score is too complicated.

Several work have solved automatic transcription for overcoming these two drawbacks. In [Takeda *et al.*, 2002], they model the problem as a hidden Markov model and do Bayesian inference to guess the most probable rhythm. In [Cemgil and Kappen, 2003], they use particle filters. In [Foscarin *et al.*, 2019], the authors address the automatic transcription problem by the mean of probabilistic context-free grammars. The interested reader may refer to the survey [Gouyon and Dixon, 2005] which is old but still informative. Automatic transcription sometimes refer to the problem of producing a music score from a raw audio signal [Benetos *et al.*, 2019].

However, automatic transcription is not about handling issues about a succinct textual source code for representing a music score.

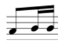
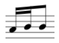
- Unlike automatic transcription, our approach – rhythm inference – takes care about partial specification, e.g. 'the 3rd note is dotted' (meaning that its duration is among 3/2, 3/4, 3/8, etc.). Note that partial specifications are handled for free with a constraint satisfaction problem formulation.
- In automatic transcription, the temporal information comes from a real performance and thus is approximately the real rhythm. In rhythm inference, it does not need to have the exact temporal information, but rather relative information (e.g. this note is likely to be longer than another note). Although the information is relative, rhythm inference is able to guess the rhythm from the positions of the measure bars.

We also note that constraint satisfaction has also been used in other context in music: expressing constraints on pitch of the note [Toro *et al.*, 2016] (whereas we express constraints on the durations of the notes).

6 Future Work

We have presented a new problem called *rhythm inference* that takes as an input a partial description of a music score and infer some consistent rhythm. Our methodology is mathematical simple: we formulate rhythm inference as a constraint satisfaction problem.

There are many ways to improve the tool and the language `abcd`. Currently, each voice is treated separately. In the future, we will also consider constraints between several voices: typically notes aligned vertically should be played at the same time. We also aim at handling properly anacrusis, i.e. measure that at generally at the beginning of a piece and whose total duration is a fraction of the signature.

We also aim at handling preferences in the constraint satisfaction problem [Dubois *et al.*, 1996] (for instance, the pattern  is preferred to  when the input is ambiguous like '`a b b `').

Acknowledgements

We thank Charlotte Truchet for some helpful discussions.

[3]https://www.abcjs.net/
[4]https://developers.google.com/optimization
[5]https://github.com/francoisschwarzentruber/abcd

References

[Benetos *et al.*, 2019] Emmanouil Benetos, Simon Dixon, Zhiyao Duan, and Sebastian Ewert. Automatic music transcription: An overview. *IEEE Signal Process. Mag.*, 36(1):20–30, 2019.

[Cemgil and Kappen, 2003] Ali Taylan Cemgil and Bert Kappen. Monte carlo methods for tempo tracking and rhythm quantization. *J. Artif. Intell. Res.*, 18:45–81, 2003.

[Dubois *et al.*, 1996] Didier Dubois, Hélène Fargier, and Henri Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Appl. Intell.*, 6(4):287–309, 1996.

[Foscarin *et al.*, 2019] Francesco Foscarin, Florent Jacquemard, and Philippe Rigaux. Modeling and learning rhythm structure. In *Sound and Music Computing Conference (SMC)*, 2019.

[Gouyon and Dixon, 2005] Fabien Gouyon and Simon Dixon. A review of automatic rhythm description systems. *Comput. Music. J.*, 29(1):34–54, 2005.

[Takeda *et al.*, 2002] Haruto Takeda, Naoki Saito, Tomoshi Otsuki, Mitsuru Nakai, Hiroshi Shimodaira, and Shigeki Sagayama. Hidden markov model for automatic transcription of MIDI signals. In *IEEE 5th Workshop on Multimedia Signal Processing, MMSP 2002, St. Thomas, Virgin Islands, USA, December 9-11, 2002*, pages 428–431. IEEE, 2002.

[Toro *et al.*, 2016] Mauricio Toro, Camilo Rueda, and Carlos Agón. Gelisp: A library to represent musical constraint satisfaction problems and search strategies. *Journal of theoretical and applied information technology*, 2016.