GENERATING TARGETED RHYTHMIC EXERCISES FOR MUSIC STUDENTS WITH CONSTRAINT SATISFACTION PROGRAMMING

Graham Percival
University of Victoria
Department of
Computer Science

Torsten Anders
University of Plymouth
Interdisciplinary Centre for
Computer Music Research

George Tzanetakis
University of Victoria
Department of
Computer Science

ABSTRACT

Generating technical exercises for various levels of playing ability is important for any instrument method book. Writing exercises by hand can be quite tedious, and severely limits the number of exercises which could be created. This is particularly apparent when we consider computer music tutoring systems, which could benefit from a library of thousands of exercises. This library could be used to pick material which addresses the specific weaknesses of students, or to ensure that the student always practices new material while working on sight-reading. We therefore turn to computer-assisted composition to generate these technical drills. This is a challenging problem for which the use of constraints provides an elegant solution.

1. INTRODUCTION

Most work in Computer-Assisted Composition (CAC) has focused on artistic goals: either replicating an existing musical style, or creating music according to a composer's desires. In this paper, we propose using CAC for pedagogical goals: creating rhythmic exercises for students.

Modelling compositional tasks with constraint programming is not new. A Constraint Satisfaction Problem (CSP) is defined as a problem composed of a finite set of variables, each of which is associated with a finite domain. A set of constraints restricts the values that the variables can simultaneously hold; the task is to assign each variable a value in its domain. A good introduction to CSPs is presented in [9]. There have been a few general CSP systems for music composition, including PWConstraints and Situation [2], OMClouds [8], and Strasheela [1]. Specific CSPs have been created; these include rhythmic patterns [7] and instrumental writing [3]. However, these efforts have focused on artistic, rather than pedagogical, goals.

Rhythmic exercises were chosen for their generality: all musicians need to learn simple rhythms. Certain musical styles and instruments may demand more advanced rhythms than others, but the basic rhythms are universal. In contrast, the difficulty of playing certain pitch combinations diverges quickly based on instrument. For example, playing two adjacent notes at once is simple on a piano, difficult on a cello, and impossible for a singer. Pedagogical exercises which include pitches would therefore be quite instrument-specific.

We begin by discussing the usefulness of computer-created exercises in section 2. This is followed by an examination of musical rhythms and how they may be divided into difficulty levels in section 3. Details of the technical implementation are explained in section 4 and discussed in section 5. We end with the conclusion and future plans in section 6.

2. APPLICATIONS

2.1. Sight-Reading Exercises

Imagine a Computer-Assisted Musical Instrument Tutoring (CAMIT) [5] program which tests the sight-reading ability of a student. It presents an exercise to the student; if the student performs the exercise correctly, she receives a high mark and is presented with a harder exercise. If she does not perform the exercise correctly, she receives a low mark and is given another exercise on the same level.

If the program only includes a few musical exercises, a student may receive high marks by memorizing those exercises. There is nothing wrong with computer testing of a prepared exercise or piece of music – but if the intent of a particular program is to test sight-reading ability, then we should ensure that the program really *is* testing sight-reading ability and not memory. To avoid repeating exercises, the CAMIT project would require dozens (if not hundreds!) of exercises for each difficulty level.

2.2. Targeted Exercises

Imagine a sophisticated CAMIT program which tests a student's rhythmic ability. In addition to analyzing the student's performance and providing a single overall grade, the program also detects specific mistakes.

For example, suppose that the student has difficulty switching between triplets and duple notes (Figure 1). Once the program detects this, it may generate exercises which use this pattern more often. Conversely, if the student has no difficulty performing this rhythm, the computer may decrease the frequency of this pattern.

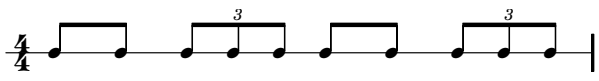


Figure 1. Triplets and duple notes.

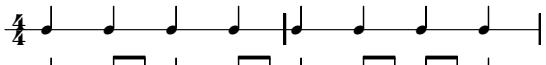
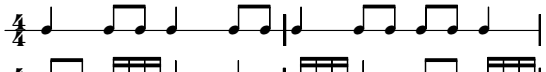
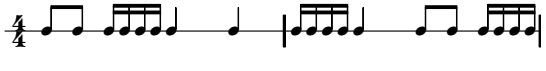
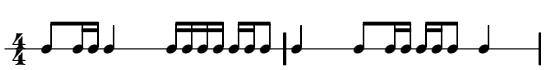
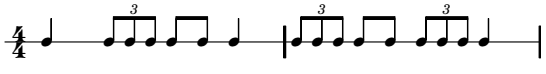
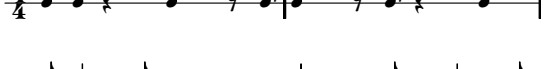
Level	Events constraints	Durs.	Example
0	Must have an event on every beat, and no rests.	$\frac{1}{4}$	
1	Must have an event on every beat, and no rests.	$\frac{1}{4}\,\frac{1}{8}$	
2	Must have an event on every beat, no rests, and each beat is divided into equal durations.	$\frac{1}{4}\,\frac{1}{8}\,\frac{1}{16}$	
3	Must have an event on every beat, no rests, and $\frac{1}{16}$ must occur in pairs replacing an $\frac{1}{8}$.	$\frac{1}{4}\,\frac{1}{8}\,\frac{1}{16}$	
4	Must have an event on every beat, no rests, and each beat is divided into equal durations.	$\frac{1}{4}\,\frac{1}{8}\,\frac{1}{12}$	
5	Must have an event on every beat, and rests can only occur on beats.	$\frac{1}{4}\,\frac{1}{8}$	
6	Must have an event on every second beat, and rests can only occur on beats.	$\frac{1}{2}\,\frac{3}{8}\,\frac{1}{4}\,\frac{1}{8}$	
...			
≈ 15	Must have an event on every second beat, and a note longer than one beat must either begin or end a beat.	$\frac{1}{2}\,\frac{7}{16}\,\frac{3}{8}\,\frac{1}{4}\,\frac{3}{16}\,\frac{1}{8}\,\frac{1}{16}$	

Table 1. Levels of rhythmic difficulty: general musical information.

Level	Beat div.	Normal solutions	Constraints to exclude uninteresting solutions	Interesting solutions
0	1	1	$\emptyset$	1
1	2	256	Cannot have three identical adjacent beats.	68
2	4	6561	Cannot have three identical adjacent beats, and must have at least eight $\frac{1}{16}$.	3167
3	4	390,625	Cannot have three identical adjacent beats, must have at least eight $\frac{1}{16}$, at least eight $\frac{1}{8}$, and at least two $\frac{1}{4}$.	8778
4	6	6561	Cannot have three identical adjacent beats, and must have at least four $\frac{1}{8}$ and six $\frac{1}{12}$.	3100
5	2	65,536	Cannot have identical durations in three adjacent beats, must have at least two rests, and cannot have adjacent rests.	6436
6	2	331,776	Must have a $\frac{1}{2}$, a $\frac{3}{8}$, a $\frac{1}{4}$ tied over a beat, and at least two rests and two notes.	5784
...				
≈ 15	4	$3.2{\cdot}10^{14}$	Must have two notes longer than $\frac{1}{4}$, at least three rests, at least eight notes, and at least eight $\frac{1}{16}$.	$> 10,000$

Table 2. Levels of rhythmic difficulty: technical details.

3. RHYTHMIC DIFFICULTY LEVELS

Most musicians have an intuitive sense of what makes a rhythm easier or harder to perform. However, we cannot teach human intuition to a computer – before we can begin creating rhythms with CAC, we must formalize our concept of "easy" and "hard" rhythms.

We begin by considering music as a series of events. An *event* represents the beginning of a note or rest, regardless of its duration (see Figure 2). If this rhythm is clapped (a very common way to practice rhythms), the two bars will sound identical. However, the second bar is harder to read, due to the quarter rest spanning the second beat. In the first bar, there is an event on each beat; this is very useful for students.

Using the terminology of events and durations, we can express different rhythmic difficulty levels quite concisely. Table 1 shows the first few levels of rhythmic difficulty. Each level is defined independently, so they may be reordered or modified if desired. Our rhythmic exercises are quite short: two bars in 4/4 time.

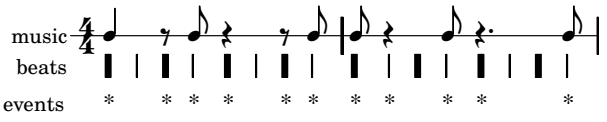


Figure 2. Sample rhythm. Beats: thick blocks show beats, while thin lines show the eighth note off-beats. Events: a star indicates an event in this position.

When we divide rhythms into difficulty levels, we must consider the readability of the exercise in addition to simply where the note onsets occur. This is quite apparent when we consider the example in level 15 – the notes which are syncopated against the beat make this exercise quite challenging (as is appropriate for this high level).

Table 2 contains technical details, including the total number of rhythms on each level. In many cases, these rhythms include rhythms which were solutions to some previous levels – for example, level 2 rhythms include all level 1 rhythms, and level 5 rhythms will not necessarily contain any rests. For some applications this may be desirable, but for other applications we may wish to generate only "interesting" rhythms. We therefore introduce extra constraints which exclude uninteresting solutions.

The "Beat div." (division) column represents how many events we divide each quarter-note beat into. For example, to represent sixteenth notes we must divide each beat into at a multiple of 4, and to represent triplet eighths we must divide each beat into a multiple of 3.

4. IMPLEMENTATION

We used Strasheela [1] and its underlying language Oz [6] to phrase rhythmic exercises as a CSP, and generated sheet music with GNU/LilyPond [4]. Oz is a multi-paradigm programming language, including functional and constraint programming.

4.1. Music Representation: Events and Durations

To express a problem as a CSP, we must state what the variables are. Most musical CSPs specify the number of notes at the outset, but this is problematic for more advanced rhythms. One beat may be filled with anywhere from zero (if this beat falls inside a long note) to eight notes (if we allow thirty-second notes).

We therefore create a list of all possible events, where each variable represents which event occurs in that position. For example, to produce two bars with no rests and notes as fast as eighth notes, we create a list of 16 variables. Each variable represents one possible event – these correspond to the "beats" in Figure 2. If there is no event at this position, the variables will be 0. The beginning of a note is represented with 1, while the beginning of a rest is represented with 2. Figure 3 shows our event list.

The event list can express any rhythm which fits into our "grid" of beats and beat divisions. However, there are some constraints which we cannot easily express, such as "must contain at least one half note". To constrain the durations directly, we introduce a second list of variables to store durations. The duration list D has the same number of variables as the event list E, and is linked to the event list so that information may pass between the two. The lists are indexed starting from position 1.

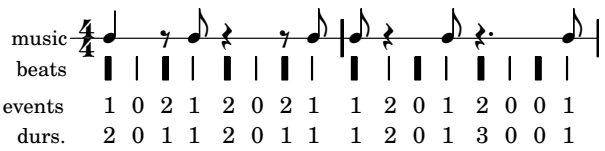


Figure 3. Sample rhythm with implementation details.

- $D_i = 0 \iff E_i = 0$
- $D_i = 1 \iff (E_i \neq 0) \wedge (E_{i+1} \neq 0)$
  e.g. starting from position i, E contains `1 1`.
- $D_i = 2 \iff (E_i \neq 0) \wedge (E_{i+1} = 0) \wedge (E_{i+2} \neq 0)$
  e.g. starting from position i, E contains `2 0 1`.
- $D_i = N, N > 2 \iff (E_i \neq 0) \wedge (E_{i+1} = 0) \wedge \ldots \wedge (E_{i+n-1} = 0) \wedge (E_{i+n} \neq 0)$
  e.g. starting from position i, E contains `1 0 0 2` (for $M = 3$).

4.2. Defining Constraints

Once we have linked our event and duration lists, implementing the constraints is straightforward – we may express constraints in terms of events and/or durations. We shall examine two levels in detail to illustrate this point.

4.2.1. Level 2 Constraints

To represent quarter notes, eighth notes, and sixteenth notes, we need to divide each beat in 4. Our event and duration lists will therefore contain 32 variables. We begin by specifying that an event occurs on every beat (1), and that we do not allow any rests (2):

$$\forall E_i | i \in \{1, 5, 9, 13, 17, 21, 25, 29\} : E_i \neq 0 \quad (1)$$

$$\forall E_i : E_i \neq 2 \quad (2)$$

Now we specify that each beat must be divided into equal durations – in other words, every non-zero duration must be equal to the first duration of that beat (3):

$$\begin{aligned} &\forall i | 0 \leq i \leq 7 : \\ &(D_{4*i+2} = 0) \quad \vee \quad (D_{4*i+2} = D_{4*i+1}) \\ &(D_{4*i+3} = 0) \quad \vee \quad (D_{4*i+3} = D_{4*i+1}) \\ &(D_{4*i+4} = 0) \quad \vee \quad (D_{4*i+4} = D_{4*i+1}) \end{aligned} \quad (3)$$

This constraint may appear to allow invalid solutions for duration sequences such as `1 0 0 1`, but such solutions are already forbidden by the linking between the Event and Duration lists.

Constraints (1), (2), and (3) define this level. We add two more constraints to ensure that we always get "interesting" exercises: we cannot have three adjacent identical beats (since the beats contain equal durations on this level, we may simply constrain the first duration of each beat (4)), and we require at least 8 sixteenth notes (5).

$$\begin{aligned} &\forall D_i | i \in \{1, 5, 9, 13, 17, 21\} : \\ &(D_i = D_{i+4}) \implies (D_{i+8} \neq D_i) \end{aligned} \quad (4)$$

$$(\sum_{D_i}^{32} D_i = 1) \geq 8 \quad (5)$$

In (5), *true* and *false* are represented by 1 and 0, allowing us to add them.

4.2.2. Level 6 Constraints

To create the durations we desire we need to divide each beat into 2, so our lists have 16 variables each. We begin by specifying that we want an event every two beats (6):

$$\forall E_i | i \in \{1, 5, 9, 13\} : E_i \neq 0 \quad (6)$$

Then we specify that rests can only occur on beats (in other words, if E_i is not a beat, it cannot be a rest (7)):

$$\forall E_i | i \in \{2, 4, 6, 8, 10, 12, 14, 16\} : E_i \neq 2 \quad (7)$$

Constraints (6) and (7) define this level. To exclude uninteresting exercises, we add four more constraints: each solution should contain at least one dotted quarter and half note (8), at least one quarter note or rest which is held over a beat (9), and contain at least two rests and two notes (10).

$$\exists D_i : D_i = 3, \quad \exists D_i : D_i = 4 \quad (8)$$

$$\exists D_i | i \in \{2, 4, 6, 8, 10, 12, 14\} : D_i = 2 \quad (9)$$

$$(\sum_{E_i}^{16} E_i = 2) \geq 2, \quad (\sum_{E_i}^{16} E_i = 1) \geq 2 \quad (10)$$

5. DISCUSSION

As previously mentioned, a CSP must specify the number of variables before searching for a solution. For our goal of creating short rhythmic exercises, it was sufficient to create a list of Events and Durations. This approach fixes the total duration (in our case, to 2 bars), but allows the number of notes to fluctuate. Another approach would be to fix the number of notes, and let the total duration vary – exercises with many sixteenth notes may only be 1 bar long, while an exercise with half notes may be 6 bars long. A more sophisticated solution would be to fix the total number of notes, but allow notes with a duration of 0. Such notes would be considered "non-existing", and would not be exported into musical notation. This method would allow both the total duration, and total number of "existing notes", to vary between different solutions. This is covered in greater detail in [1].

The simple rhythms produced in our exercises do not use any tied notes. However, our music representation may be easily extended to create rhythms which include ties: let 3 in the event list represent "begin a note which is tied to the previous note". We also need the constraint that 3 cannot follow 2 (you cannot tie a rest to a note). With those small modifications, we can begin writing constraints to allow ties in any position we desire.

The rhythms in real music generally do not change as quickly as our exercises: a single rhythmic pattern may be repeated for many bars as a time, especially for an instrument which is not playing a melody. Our CSP could easily be modified to produce such rhythms: create an exercise which is 8 bars long, but add the constraint that the first two beats are repeated 16 times.

6. CONCLUSION AND FUTURE WORK

We have presented a method of using CSPs to create rhythms of specific difficulty levels. Rhythmic difficulty levels are easily expressed by constraints, and the resulting exercises may be easily integrated into computer-assisted musical instrument tutoring programs. Formulating constraints in terms of linked events and durations allows us to express rhythmic difficulty in easily-understandable musical terms.

We chose to begin with rhythms since they may benefit all musicians. Exercises with pitches are quite instrument-dependent, so they only benefit those specific instrument(s). Our future work will focus on creating pitched exercises for violin and cello.

7. REFERENCES

[1] T. Anders. Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System. Ph.D. Thesis, School of Music & Sonic Arts, Queen's University Belfast, 2007.

[2] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue. Computer-Assisted Composition At IRCAM: From PatchWork to OpenMusic. *Computer Music Journal,* 23(3):59–72, MIT Press, Cambridge, MA, USA, 1999.

[3] M. Laurson and M. Kuuskankare. A Constraint Based Approach to Musical Textures and Instrumental Writing. *CP01 workshop on Musical Constraints,* Cyprus, 2001.

[4] H. Nienhuys, J. Nieuwenhuizen, and G. Percival. LilyPond Music Notation. http://www.lilypond.org/, February 2008.

[5] G. Percival, Y. Wang, and G. Tzanetakis. Effective Use of Multimedia for Computer-Assisted Musical Instrument Tutoring. *EMME '07: Proceedings of the international workshop on Educational Multimedia and Multimedia Education,* pages 67–76, Augsburg, Bavaria, Germany, 2007.

[6] P. Van Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming.* The MIT Press, March 2004.

[7] O. Sandred. Searching for a Rhythmical Language. *PRISMA 01,* EuresisEdizioni, Milano, 2003.

[8] C. Truchet, G. Assayag, and P. Codognet. OMClouds, a Heuristic Solver for Musical Constraints. *MIC2003: The Fifth Metaheuristics International Conference,* 2003.

[9] E.P.K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London and San Diego, Out of print; available for download at http://www.bracil.net/edward/FCS.html, 1993.