Towards Real-time Score Analysis in PWGL

Mika Kuuskankare[1]

[1]Independent grant researcher

mkuuskan@yahoo.com

Abstract. *In this paper, we introduce an original approach to computerized music analysis within the graphical computer-assisted composition environment called PWGL. Our aim is to facilitate real-time analysis of interactive scores written in common Western music notation. To this end, we have developed a novel library that allows us to analyze scores realized with the help of ENP (the graphical music notation module of PWGL), and to visualize the results of the analysis in real-time. ENP is extended to support the display of supplementary information that can be drawn on top of the score as an overlay. The analysis backend is realized with the help of our built-in musical scripting language based on pattern matching. The analysis results are presented directly as a part of the original score leveraging the extensible and interactive visualization capabilities of ENP.*
In this paper we describe the current state of the library and present, as a case study, a fully functional application, which allows for the real-time analysis and display of voice leading errors according to the counterpoint rules developed mainly in the Renaissance and Baroque eras.

1. Introduction

Typically, computational musicology employs *non-real-time* algorithms to analyze music encoded in textual form. Within the computer music community, *real-time music analysis* has become somewhat synonymous to real-time audio analysis and visualization. In this paper, we focus on *analysis of interactive scores written in common Western music notation*, an approach still underrepresented within the community mainly because of the complex nature of computerized music notation.

It is customary to divide real-time systems into two main categories: hard real-time and soft real-time. Systems falling under the first category usually present either safety or mission critical constraints and, in general, missing a deadline is considered disastrous. Systems in the second category work ”fast enough” and missed deadlines affect only the quality of service. This article won’t debate the fine points of real-time computing. Instead, for the purposes of this paper, we consider the latter category, specifically in the sense of fast–or fast enough–computation.

Encapsulating the notational and analytical practices of the western art music in a real-time context, requires versatile and interactive representations of music, flexible visualization devices, and musical knowledge encoded into the system. Let us consider the representation of music in state-of-the-art real-time and non-real-time musical environments. *Real-time* environments, such as Pure Data[1], concentrate on producing and manipulating sound, video, and graphics, rarely providing robust interactive representations of musical notation because of the computational complexity associated with these operations. *Non-real-time* environments, such as OpenMusic[2] for computer-assisted composition or LilyPond[3] for music notation, allow for robust musical representations at the expense of interactivity.

Rich music analytical representations and real-time interactivity need not to be mutually exclusive, instead, the two should complement each other. Text editors, for example, allow people to develop documents in real-time while retaining a rich view of these documents. The same is true of photo editing, where users can apply sophisticated filters to their photos and instantly see the results.

PWGL[4] can already be used to analyze musical scores realized with the help ENP[5] using a variety of methods[6, 7, 8, 9]. However, the existing methods have not yet been adapted for real-time analysis. In this paper, we present a novel PWGL library, that aims to make it possible to analyze ENP scores and to visualize the results of the analysis in real-time. The analysis backend is realized using ENP-Script[10].

PWGL is a visual music programming language written in Lisp and OpenGL. Its primary focus is on computer-assisted composition in an integrated environment with music notation and software synthesis. ENP, in turn, is PWGL’s built-in music notation program developed primarily for applications concerning computer-assisted composition and virtual instrument control. It is designed to produce automatic, reasonable quality musical typesetting. Finally, ENP-Script is the native scripting language of ENP. It allows us to access complex musical patterns with the help of a pattern-matching syntax developed for our constraint-based compositional system.

The rest of the paper is organized as follows. First, we introduce the two key software components of the library: ENP and ENP-Script. Next, we discuss implementation details and present an application prototype that allows for real-time analysis and display of common voice leading errors. The paper ends with some concluding remarks and ideas for future work.

2. ENP

ENP is a full-featured music notation program with a graphical user-interface and an extensive set of graphical devices for enriching notational output. ENP can produce relatively complex, automatically typeset and interactive music notation (see Figure 3).

At the center of the expressive power of ENP are interactive and user-definable graphical devices, called *expressions*. In ENP, the term expression is used in a broader sense than as a traditional expression marking. ENP-Expressions are multipurpose visualization devices that can be used to represent complex dynamic Lisp-based objects as a part of a musical texture (see Figure 4). In addition to their traditional use, they can be used in a wide range of applications, such as providing control information for virtual instrument synthesis. One of the unique features of ENP-Expressions is that they can be attached to a discontinuous group of objects spanning across multiple parts. This is especially beneficial in musical analysis applications that typically deal with the relationships between independent melodic lines and simultaneously sounding harmonies.

3. ENP-Script

ENP-Script is the scripting language of ENP. It uses the syntax of the pattern-matching language of PWConstraints[11]. PWConstraints, in turn, is a compositional system that uses the Backtracking algorithm to obtain a solution to musical constraint satisfaction problems. ENP-Script, while sharing most of its behavior with PW-Constraints (e.g., language syntax, score access), does not use backtracking. The scripts consist of one or more scripting rules. The scripting rule, in turn, consists of a pattern-matching part and a Lisp-code part. The pattern-matching part contains a set of pattern-matching variables each representing an object in the score. Pattern-matching variables hold information about the pitch of the associated notes, rhythm, current harmony, etc. Once the pattern-matching part of a rule has accessed the desired objects (i.e., notes, chords, measures, harmonic formations), these variables can be used in the Lisp-code part. To access information from the variables, the `'m'` method is typically used ('m' standing for multi-accessor). Normally, the `m` method returns either a single note for non-compound objects or a list of notes for compound objects (e.g., chords, beats, measures, harmonic formations). It also accepts a list of keyword arguments that allow more precise specification of the type of data that will be returned. Finally, the scripting engine loops through the score's notational objects and applies the scripting rules, usually to produce some effect, such as attaching an expression. For more detailed discussion of ENP-Script, refer to reference [10].

4. The Library

The present library contains two modules: (1) the *visualization module*, and (2) the *analysis module*. The analyze-visualize cycle is triggered every time a property of the score is changed. This typically happens when the user, for example, transposes a note.

4.1. The Visualization Module

The results of the analysis are visualized directly on the source score. For the purposes of the present library, a new category of ENP-Expressions–called *overlay-expression*–was developed. Overlay-expressions differ from standard ones in a few specific ways: (1) they are not part of the

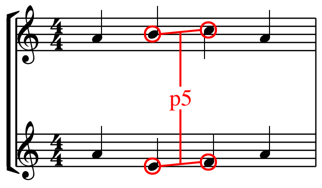

Figure 1: The voice-leading expression indicating the notes involved as well as the name of the error ('P5' indicates a parallel).

object structure of the score, i.e., they are not saved with the associated score, and (2) they are temporary, i.e., the lifetime of the overlay objects ends before the next batch of objects begins to be initialized. Instead, as the name suggests, they are drawn as an overlay on top of existing score objects. Most expressions, such as dynamics or articulations, are usually attached to objects in one single part, therefore relying on the local coordinate system of that part. However, an overlay-expression refers to the global positions of the objects it is associated with and, therefore, can easily extend itself across different parts (as can be seen in Figure 1).

The library implements two expressions that extend the overlay-expression. These are called the *voice-leading expression* and the *voicing expression*. Figure 1 shows the voice-leading expression. It holds a reference to a pair of notes in two different parts. Visually, it connects the notes in each part with a line that has a hollow circle on each end. A vertical line connects the note pairs involved in the offending voice leading case and displays an error code (e.g., 'P5' for parallel fifths) around the midpoint of the line segment. The voicing-expression, in turn, marks cases in the score where there is either a voice crossing, or an open spacing. The voicing-expression holds a reference to the corresponding notes that–by definition–have to be in two different parts. The two notes are encircled and connected vertically with a dotted line. The name of the error is again indicated around the midpoint of the line segment.

4.2. The Analysis Module

The analysis module is implemented in two parts: (1) as a real-time analysis engine, and (2) as a set of analysis rules.

The real-time analysis engine implements a slightly modified version of our scripting engine. One of the major differences is that the additional structures (see Figure 2) prepared in advance to assist in accessing the multitude of different score objects (notes, harmonies, etc.) are cached instead of being regenerated every time a script is run. One of the most complex of these structures is the harmonic structure shown in Figure 2. As a further optimization, the scripts are also only applied to the part of the score that is visible to the user.

The complete set of rules implemented by the library currently contains more than 10 rules. For the purposes of this paper, we consider only the following

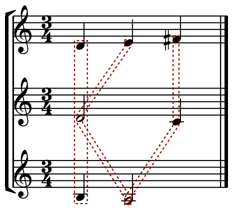

Figure 2: To assist and optimize the search process, several auxiliary structures are generated for both our constraints and scripting engines. The simultaneity (i.e, harmonic) structure is shown here. The polygons enclose the three possible simultaneities in this particular score.

rules: (1) parallel fifths and octaves, (2) voice crossing, and (3) open spacing. Parallel fifths and octaves are progressions in which the interval of a perfect fifth or an octave is followed by the same interval between the same two parts. Voice crossing is the intersection of melodic lines, e.g., a lower voice crosses above a higher one. Spacing, refers to the simultaneous vertical placement of notes in relation to each other. Open spacing typically occurs when the interval between consecutive voices exceeds an octave. Note, that the details about the interpretation of different rules typically varies from period to period.

Next, we will examine in detail the code for detecting parallel fifths progressions shown in Listing 1. Line 1 of Listing 1, defines the pattern and type of score objects (indicated by the keyword :harmony) to which the rule is applied. The pattern consists of a star (`'*'`) which matches zero or more score objects and a variable name (`'?1'`). This particular pattern extracts all the harmonic formations (groups of simultaneously sounding notes) from the score and binds them one by one to the variable named `'?1'`. In line 3, we call the m-method with the keyword `':vl-matrix'` as an argument to return an object of type *vl-matrix* (voice leading matrix). The vl-matrix object encapsulates the relevant notes that participate in all the possible voice leading cases between consecutive simultaneities. In four part texture this results in at most six different cases: voice 1 against voices 2, 3, and 4; voice 2 against voices 3 and 4; and voice 3 against voice 4. The presence of a rest in any of the voices naturally reduces the number of combinations. In line 5, we access the vl-matrix object to retrieve a list of notes belonging to the top-most voice. Next, we check that the voice actually moves up or down, otherwise this voice will not contribute to any parallel movement. Next, we access the pitch properties of the two notes and check if they are equal (line 6). In case the pitches are not equal, we need to loop through all the remaining voices to check for parallel fifths (line 7). For each voice, we again make sure that it is moving either up or down (line 9). If so, we proceed with the remaining checks. Lines 10-11 check that intervals between the two voices are both perfect fifths, while lines 12-13, in turn, check that both voices also move in the *same* direction.

If all the aforementioned conditions are satisfied, the two voices move in parallel fifths. To indicate this in the score, we instantiate an overlay-expression that is subsequently attached to the formation of the four notes (two notes in each of the involved parts). This is done in lines 14-17. The overlay-expressions have many user-definable properties, one of which is color. Here, we chose to display the overlay in red (see line 17) to denote a high level of urgency. The actual drawing of the overlays is delayed until all the voice leading rules have been checked.

5. The Application

As a proof of concept, we have built a fully functional application that allows for the analysis and display of common voice-leading errors. The application is shown in Figure 5. The current set of available rules is displayed on the right as checkboxes. These GUI components can be used to activate or de-activate rules. Only errors selected by the user are checked. The related visualizations are updated in real-time as the user works on the score. In addition to pitch, the user can manipulate any other score property, including rhythm.

6. Performance Measures

We collected some preliminary performance measures with the help of our application prototype. The tests were performed on a 2012 Apple MacBook Pro with an Intel Core i7 processor running at 2.7 GHz. The results are presented in Table 1. The 'total' time includes the initial layout calculation and drawing of the score ($\approx$ 10 ms), the analysis phase ($\approx$ 15 ms) and the drawing of any overlay objects ($\approx$ 2 ms). The response time of slightly under 30 ms seems to be in the realm of real-time and the GUI response confirms this. Thus, our experiment aligns well with our initial objectives.

Step	**Time (ms)**
layout/draw	10
analyze	15[1]
visualize	2
total	27

Table 1: The approximate timings of the steps involved in the analysis process.
[1]$\approx$ 35 ms without caching

7. Acknowledgements

This work has been supported by the Jenny and Antti Wihuri Foundation.

8. Conclusions

In this paper we presented a novel PWGL library that allows for the real-time analysis of musical scores written in the ENP score format. The potential application fields range from elementary music pedagogy to advanced counterpoint exercises, as well as computer-assisted composition. Our work could be applied, for example, in computer-based training, making it possible to develop interactive

Listing 1: Code for detecting parallel fifth progressions in an ENP score

```
(* ?1 :harmony
 (?if
  (let ((mat (matrix-access (m ?1 :vl-matrix 2 :object t) :h)))
    (when mat
      (destructuring-bind (upper-1 upper-2) (first mat)
        (unless (= (m upper-1) (m upper-2))
          (loop for mel2 in (rest mat) do
                (destructuring-bind (lower-1 lower-2) mel2
                  (unless (/= (m lower-1) (m lower-2))
                    (when (and (= (mod12 (abs (- (m upper-1) (m lower-1)))) 7)
                               (= (mod12 (abs (- (m upper-2) (m lower-2)))) 7)
                               (= (signum (- (m upper-2) (m upper-1)))
                                  (signum (- (m lower-2) (m lower-1)))))
                      (add-overlay upper-1 upper-2 lower-1 lower-2
                                   :kind :voice-leading
                                   :id :parallel-fifth
                                   :color :red))))))))))))
```

music theory applications. One possible application could be an advanced counterpoint tutor with built-in knowledge of common voice-leading and harmony rules. It would display the assignments in music notation, and correct and instruct students in real-time using a combination of text, colors, and commonly accepted music theoretical markings. Being able to see the analysis in real-time can be an effective way to gain understanding of how different elements of the composition (e.g., harmony, melody) work together. Furthermore, our tool could be used not only as a music analysis tool, but also as a compositional tool. Different kinds of rules, such as those keeping track of harmonic or melodic pitch class sets, could be used as visual reminders. Additionally, more sophisticated rules (e.g., checking the correctness of Harp pedaling) could be used to assist in the compositional process to allow the composer to focus on other tasks. Finally, the user could also invent their own expressions, based on the overlay-expression, to fit their particular needs.

Currently, the library is under active development. There will undoubtably be ample opportunities for performance optimizations, such as precompiling and caching the analysis rules themselves. We are also working on additional analysis backends, one of which will be based on the Humdrum Toolkit[12]. The Humdrum Toolkit is a widely-used open-source software package for musicological research developed at CCARH at Stanford University. It would allow us to access the rich and readily available music analysis features of Humdrum and a vast number of freely available pieces encoded in the kern data format. The different backends would not be mutually exclusive; instead, they would complement each other.

References

[1] Miller Puckette. Using Pd as a score language. In *Proceedings of International Computer Music Conference*, pages 184–187, 2002.

[2] Jean Bresson and Carlos Agon. Visual programming and music score generation with openmusic. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2011.

[3] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In *XIV Colloquium on Musical Informatics (XIV CIM 2003)*, Firenze, Italy, 2003.

[4] Mikael Laurson, Mika Kuuskankare, and Vesa Norilo. An Overview of PWGL, a Visual Programming Environment for Music. *Computer Music Journal*, 33(1):19–31, 2009.

[5] Mika Kuuskankare and Mikael Laurson. Expressive Notation Package. *Computer Music Journal*, 30(4):67–79, 2006.

[6] Mikael Laurson, Mika Kuuskankare, and Kimmo Kuitunen. Introduction to computer-assited music analysis in PWGL. In *Sound and Music Computing '05*, 2005.

[7] Mikael Laurson, Mika Kuuskankare, and Kimmo Kuitunen. The Visualisation of Computer-assisted Music Analysis Information in PWGL. *Journal of New Music Research*, 37(1):61–76, 2008.

[8] Mika Kuuskankare. Schenkerian analysis tools in ENP. In *Proceedings International Computer Music Conference*, 2013.

[9] Mika Kuuskankare and Craig Sapp. Visual Humdrum-library for PWGL. In *Proceedings of ISMIR*, 2013.

[10] Mika Kuuskankare and Mikael Laurson. Intelligent Scripting in ENP using PWConstraints. In *Proceedings of International Computer Music Conference*, pages 684–687, Miami, USA, 2004.

[11] Mikael Laurson. PWConstraints. In G. Haus and I. Pighi, editors, *X Colloquio di Informatica Musicale*, pages 332–335, Milano, 1993. Associazione di Informatica Musicale Italiana.

[12] David Huron. Music information processing using the Humdrum Toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(2):15–30, 2002.

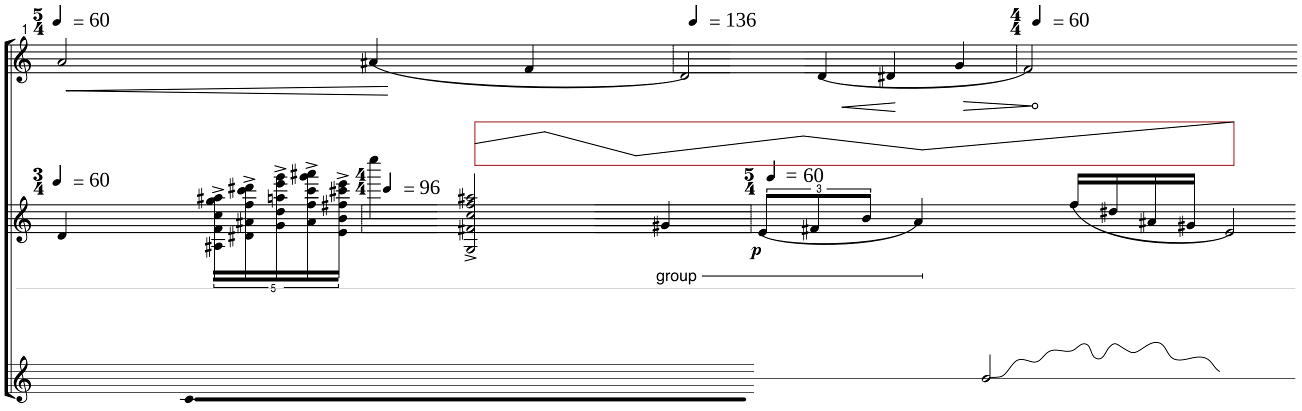

Figure 3: ***A relatively complex score notated with the help of ENP. Besides standard notational devices, such as articulations and dynamics, the score showcases several types of graphical control information as well as different simultaneous tempi and time signatures.***

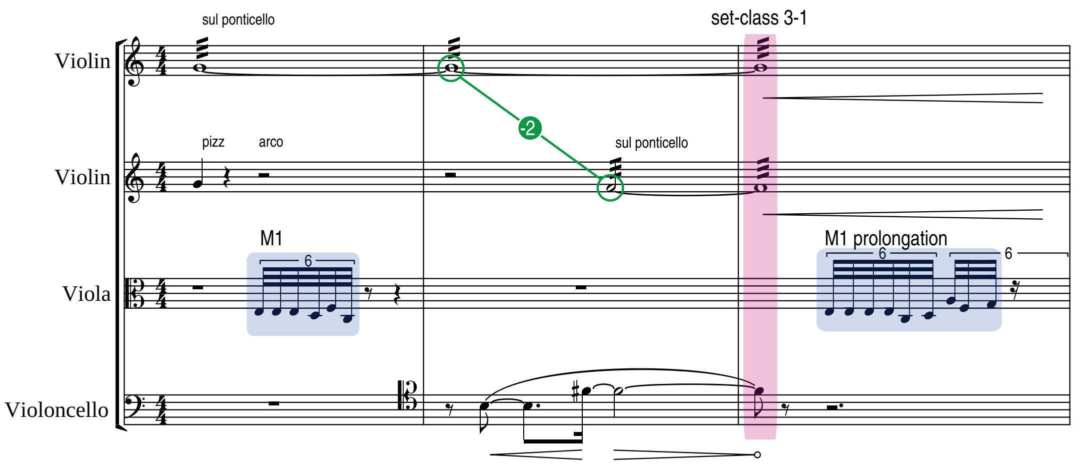

Figure 4: ***An example of different kinds of analytical markings. Here, we use a combination of text and colored areas to highlight motives as well as commonly accepted music theoretical markings, such as set class names to display the total harmonic content of selected simultaneities.***

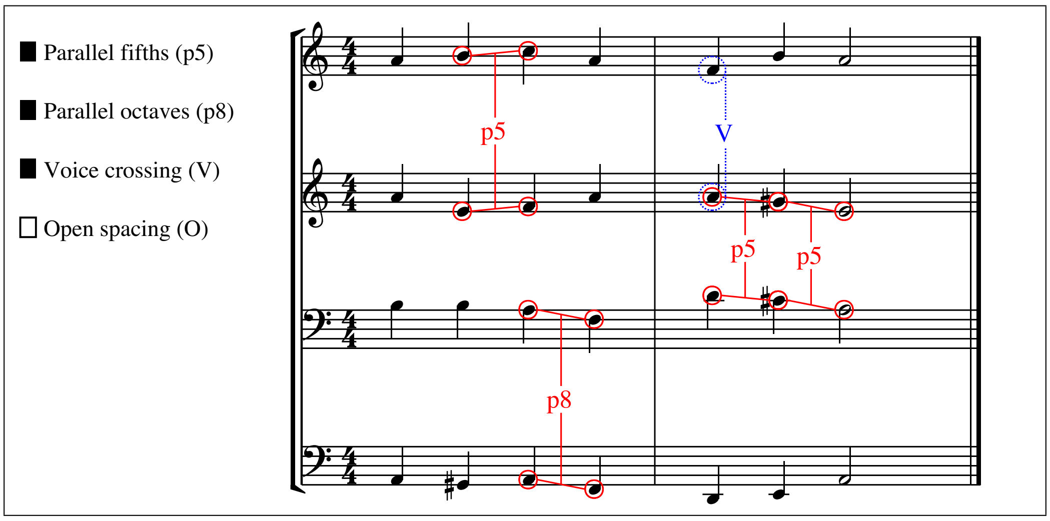

Figure 5: ***The application displaying common voice-leading errors.***