TOWARDS MUSICDIFF: A FOUNDATION FOR IMPROVED OPTICAL MUSIC RECOGNITION USING MULTIPLE RECOGNIZERS

Ian Knopke
Music Informatics Program
School of Informatics
Indiana University
iknopke@indiana.edu

Donald Byrd
School of Informatics &
Jacobs School of Music
Indiana University
donbyrd@indiana.edu

ABSTRACT

This paper presents work towards a "musicdiff" program for comparing files representing different versions of the same piece, primarily in the context of comparing versions produced by different optical music recognition (OMR) programs. Previous work by the current authors and others strongly suggests that using multiple recognizers will make it possible to improve OMR accuracy substantially. The basic methodology requires several stages: documents must be scanned and submitted to several OMR programs, programs whose strengths and weaknesses have previously been evaluated in detail. We discuss techniques we have implemented for normalization, alignment and rudimentary error correction. We also describe a visualization tool for comparing multiple versions on a measure-by-measure basis.

1 INTRODUCTION

This paper describes work on "musicdiff" program for music notation. There are many potential applications for such a program, just as there are for the text-file comparison programs that are ubiquitous these days. However, the one we are most interested in here is for use in an "engine" for a multiple-recognizer Optical Music Recognition (MROMR) system of the type described by Byrd and Schindele (2006). In fact, their work was part of a feasibility study for our MeTAMuSE project.

A large part of the musicdiff problem is alignment, where the musical documents to be aligned can be expected to be relatively similar. In particular, they should have exactly the same structure, except in terms of details. So, if sections A and B in one version appear in reverse order in the other, ideally, the program would detect that; but if it doesn't and instead says a section A before B was deleted and a new section (in fact, A) was inserted after it, it is not a serious problem.

The first well-known UNIX diff program dates back to the seventies (Hunt and McIlroy, 1976), and many variations have been created since. Some do line-level comparisons, some do character-level, and some, e.g., Microsoft Word, do both. The rough analogue of lines in this context is probably measures. But a measure can contain a lot of information. For direct use by people, something with finer granularity is highly desirable; for use in a MROMR engine, it is absolutely essential.

For MROMR, we need to compare music-notation files generated by different programs. As of this writing, MusicXML is far and away the best choice because of its widespread support: we will soon say more about this.

While we are most interested in a fully-automatic musicdiff as a component of an MROMR engine, we believe it also has great promise for interactive use by musicians. We shall say more about this application at the end of the paper.

2 MUSICXML AS AN OMR COMPARISON FORMAT

Clearly a diff program requires its input files to be in formats it can compare. Until the recent widespread adoption of MusicXML for programs handling music notation, in practical terms, such formats did not exist. Now, with an intermediate conversion step, it is possible to write MusicXML from just about any OMR program, and to read it for editing or error-checking with most notation programs.

XML is rapidly becoming one of the most prominent formats for data interchange, especially in the context of the Internet. An XML document can specify a format it uses; document formats are defined by specifications like the Document Type Definition (DTD). A DTD specifies the available element types, the relationships between them, and the attributes and ranges of values for elements and attributes.

MusicXML is an XML document format, defined by its DTD. It takes its inspiration primarily from the MuseData (Selfridge-Field, 3 4) and Humdrum music encodings (Huron, 1997). MusicXML seeks to create a common interchange format between programs that use symbolic music data. The adoption of MusicXML by many popular notation programs has accelerated its usage significantly, to the point where it is showing signs of becoming a de facto standard.

The nested-element layout of XML files is best represented as a tree data structure. Tree structures and tree

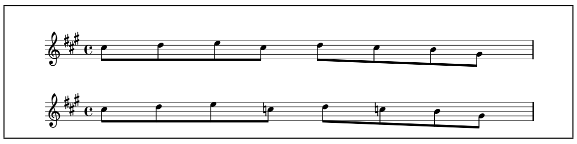

Figure 1. Correct and Incorrect Encoding

comparisons are one of the most-studied problems in computer science, so it would seem on the surface that creating a "diff" program for MusicXML documents would be easy, especially since there has been a fair amount of research specifically on comparing XML (Cobena et al., 2001). However, this assumption is incorrect. In all, three types of comparison difficulties can be identified:

1. Stylistic differences in MusicXML coding
2. Errors as a product of the OMR procedure
3. Errors in interpreting the MusicXML standard

MusicXML is fairly well-defined, but does allow considerable latitude in the use of optional elements and attributes. These differences in encoding "style" make direct comparison of XML trees unworkable. For instance, MusicXML has MIDI elements, but they may or may not be included in a file, regardless of the notational information. In effect, overlaid on every MusicXML file is a second structure that in many situations does not map especially well onto the underlying XML. Another problem is that many XML tools, including the standard XSLT tools, are based around a stateless data processing model, which may be difficult or impossible to use for many MusicXML tasks, such as those involving its extremely general "forward" and "backward" elements, or items such as slurs that stretch between branches.

A more subtle set of errors occurs in some MusicXML files from misinterpretations of the MusicXML DTD. One of the most alarming examples occurred in the case of one program writing pitches as they are notated, instead of how they sound as MusicXML requires. This would result in measures where the accidental of a note is indicated on the first occurrence of a pitch, but is incorrect throughout the remaineder of the measure. Figure 1 shows both the correct and incorrect encodings. While this problem has been fixed in the most recent version of this program, other interpretational errors remain. An ad-hoc survey of MusicXML documents produced by other programs, including many open source programs, revealed a host of other interpretational issues. In defense of MusicXML, it should be noted that its authors support an active mailing list and are working to resolve many such problems within the music notation community as a whole, and there have been many improvements even within the life of this project. However, anyone seeking to use MusicXML as an interchange format at the time of this writing should resign themselves to a certain amount of experimenting. As an indication of the extent of the problem, counts of certain elements from the feasibility study document collection are given in Table 1; note the many discrepancies.

Table 1. MusicXML element counts from the feasibility study documents

Element	PhotoScore	SharpEye	SmartScore
accidental	394	452	471
alter	260	1400	1432
backup	16	6	32
barline	28	30	21
beam	6809	6937	6485
duration	5530	5588	5396
dynamics	85	165	121
fermata	0	5	18
fifths	153	30	36
forward	1	13	24
key	153	30	36
measure	869	868	895
midi-instrument	20	0	21
pitch	5016	5158	4888
slur	1105	1366	846
stem	5016	5106	4843
tie	132	136	226
tuplet	48	8	2

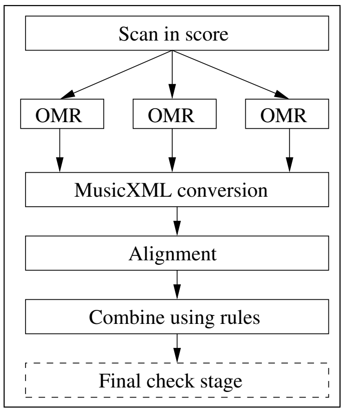

Figure 2. Scanning and document correction procedure

3 METHODOLOGY

Our methodology is based on the assumption that different OMR programs have different strengths and weaknesses, and that we can determine what they are and take advantage of that knowledge to improve recognition rates. The basic procedure is described in Figure 2. The process can be described as follows: a score is scanned in at a suitable resolution, normally 300 dpi, and then converted to MusicXML using several different OMR programs. The results are normalized and aligned at the measure level. All versions of each measure are then compared, and rules for combining them so as to minimize errors are applied, based on the situation. Finally, an optional check stage may be used; this could involve human checking or an automated heuristic of some sort.

At the moment, the procedure of scanning scores is being done by hand. However, we are experimenting with code for mouse gesture control that, in combination with some other scripting, aims to automate the entire process.

The core of the process is "combining using rules", the error correction stage. This requires an "error map" of the strengths and weaknesses of each program; such maps were manually created, albeit in a rudimentary form, in the original feasibility study for this project (Byrd and Schindele, 2006). Unfortunately, ours is a "moving target": new versions of programs appear regularly, and different combinations of programs may be available to different users of the final system. What is required is a system for *automatically* creating such an error map. Our approach to this is to use a set of "regression" test files for which we have both scans and ground truth files. A comparison of what the output should be and what it is for these files can then be used to map out likely errors.

For these purposes, two test collections have been created. The first is the original collection of pieces from the feasibility study by Byrd and Schindele (2006). Most of these are normal selections from the classical canon, and as such have many articulations, editorial marks, rehearsal numbers, and other markings that occur in common practice notation. However, the presence of these additional symbols makes the job of the OMR programs more difficult because of the increased opportunities for confusion between symbols. Consequently, for developmental purposes, we were interested in creating additional test material with very little other than notes, barlines, clefs, and key signatures. As an initial step, a second data set was created from a collection of unaccompanied recorder music (Van Eyck, 1986) consisting of 62 brief pieces. Three different OMR programs were used for these tests: PhotoScore, SharpEye, and SmartScore. These are occasionally designated in the text as **ps**, **se**, and **ss** respectively.

3.1 Normalization

Normalization is required because MusicXML supports multiple ways to encode the same content. Our normalization stage consists primarily of converting all durations to a common timebase. We also make default values explicit, and we put simultaneous MusicXML elements like notes in a chord into a standard order.

3.2 Alignment

Many of the OMR errors we have seen are missing barlines, or misinterpreting them for other vertical structures like note stems or vice-versa. The result is missing or extra measures, a serious problem because proper alignment of measures is essential before any kind of comparison can be undertaken.

Our first attempt to solve this problem computed a number summarizing the content of each measure, then used these numbers to find the proper alignment of measures. While partially successful, we found cases where each program made slightly different mistakes in a series of measures, producing a string of slightly different scores; as a result, the algorithm tends to get lost and produce improper alignments.

Our current alignment system uses edit distance with a dynamic programming algorithm (Durbin et al., 1998; Gusfield, 1997) that is heavily weighted to preserving measure alignment. This is similar to many other systems used for computing music similarity (Smith et al., 1998), except that we align three (or perhaps more) note streams at the same time. Instead of the normal two dimensions, here the alignment for both the local and global matrices occurs in a multi-dimensional space, with the output of each program assigned to a different axis.

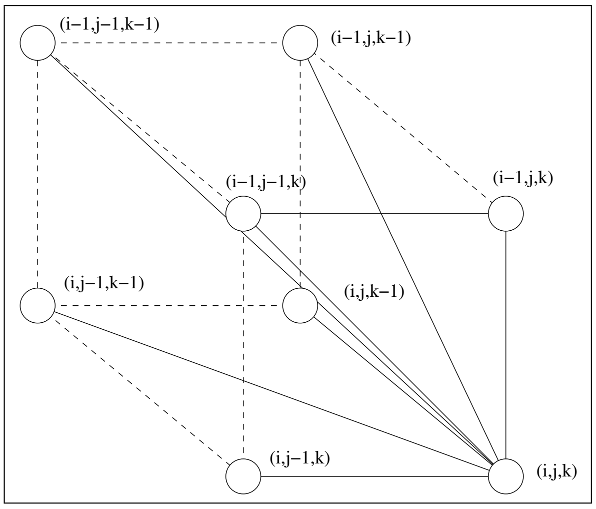

Figure 3. Three-dimensional global scoring "cube"

Each value in the local score matrix is computed as follows:

$$p(i,j,k) = [avg(p_i + p_j + p_k) - min(p_i + p_j + p_k)] \quad (1)$$

$$d(i,j,k) = [avg(d_i + d_j + d_k) - min(d_i + d_j + d_k)] \quad (2)$$

$$l(i,j,k) = q \times p(i,j,k) + (1-q) \times d(i,j,k); \quad (3)$$

where p is pitch and d is duration. q is an adjustment factor that controls the weighting between the pitch and duration components: a value of 0 pays attention to pitch alone, 1 to duration alone.

A two-dimensional dynamic programming algorithm considers the insertion, deletion, and substitution costs between any pair of notes in the two melodies. Each step can be thought of as involving four points at the corners of a square, producing an answer in the lower right from scores calculated for the other three. In the three-dimensional case, the square becomes a cube, requiring the calculation of seven scores instead of three, as shown in Figure 3. Additionally, we assign a high cost to substitutions between a barline and a note, which essentially prevents note/barline mismatches. For each token, this provides a collection of eight values; the minimum is taken and assigned to the global score matrix.

This algorithm gives significantly better results in our tests so far. In practice, we have found that higher q values tend to produce better results. This appears to be because mistakes in pitch (besides missing accidentals) are less common than duration errors with OMR; furthermore, in our experience, pitch errors are almost always the result of missing notes entirely, which also produces duration errors.

One problem with the above algorithm is that each additional stream to align doubles the amount of work needed, so it does not scale well in this respect. However, it should still perform acceptably with, say, five versions, which is probably near the practical limit anyway.

3.3 Error Correction

Once proper measure alignment has been achieved, corrective rules must be applied on a case by case basis to any errors. For this purpose, we define an error as any situation where the the programs do not all agree.

Errors can be corrected in one of several ways. The program can simply decide to disregard the output of some of the programs in that circumstance. In situations where all programs disagree the context becomes more important, and actual corrective code must be applied.

There is much work to do in this area, and this system is still quite rudimentary. Nevertheless, our initial experiments show that even a small set of corrections and selection procedures can greatly improve the quality of the output.

3.4 Document Visualization and Non-OMR Applications

We have created a tool that can convert multiple aligned MusicXML fragments to traditional notation for easy visual inspection. The output format is images embedded in a standard HTML page stored on a web server. Our tool also provides MIDI playback and offers access to the MusicXML used to produce the notation.

Turning to the question of using music interactively rather than as part of an MROMR engine, the work on an "Electronic Variorum Edition" of Don Quixote described by Kochumman et al. (2004) might serve as a model. They have written a standalone multisource editor (designed for scholars) and a web-based "virtual edition" viewer (for readers). In music, as far as we know, nothing similar has been done with symbolic comparison, but quite a bit via superimposing one (semi-transparent) image on a similar one and letting the user find any differences. For example, the Online Chopin Variorium Edition OCVE (2007) is relying solely on image superimposition so far, but their pilot-project final report shows (and discussion with principals of the project supports this) that they appreciate the advantages of symbolic comparison. Unfortunately, properly comparing music as complex as Chopin's is well beyond what we expect to accomplish in the near future.

4 CONCLUSIONS

At present, we have a prototype system that uses multiple OMR recognizers to produce a composite document that is superior to the output of any one of the programs. While it has not been tested much thus far, we believe it has the potential to lead to considerably better OMR systems. As we were completing this paper, we learned of a new version of PhotoScore ("Ultimate 5") that apparently utilizes two recognizers. This both supports our intuitions about MROMR and gives us an opportunity to evaluate the concept in a context more limited than what we envision.

5 ACKNOWLEDGEMENTS

We would like to acknowledge the support of Don Waters and Suzanne Lodato the Andrew W. Mellon Foundation. Without their support this work would have been difficult, if not impossible. We would also like to thank Geraint Wiggins, Tim Crawford, and Chris Raphael for their help, conversation, and the occasional brain transplant. Additional thanks to Bill Guerin for technical support.

References

Byrd, D. and Schindele, M. (2006). Prospects for improving optical music recognition with multiple recognizers. In *Proceedings of the International Conference on Music Information Retrieval*, pages 41–6.

Cobena, G., Abiteboul, S., and Marian, A. (2001). Detecting changes in XML documents. In *BDA*.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

Hunt, J. W. and McIlroy, M. (1976). An algorithm for differential file comparison. Technical Report 41, Computing Science Technical Report, Bell Laboratories.

Huron, D. (1997). *Humdrum and Kern: Selective feature encoding*, pages 375–401. MIT Press, Cambridge, Mass.

Kochumman, R., Monroy, C., Deng, J., Furuta, R., and Urbina, E. (2004). Tools for a new generation of scholarly edition unified by a TEI-based interchange format. In *Proceedings of the 2004 Joint ACM/IEEE Conference on Digital Libraries*, pages 368–9.

OCVE (2007). Online Chopin Variorum Edition. `http://www.ocve.org.uk/`.

Selfridge-Field, E. (1993–4). The MuseData universe: A system of musical information. *Computing in Musicology*, (9):9–30.

Smith, L., McNab, R. J., and Witten, I. H. (1998). Sequence-based melodic comparison: A dynamic-programming approach. *Computing in Musicology*, (11):101–18.

Van Eyck (1986). Der Fluyten Lust hof, Volume I. New Vellekoop Edition.