On The Analysis of Musical Performance by Computer

Douglas McGilvray

October 23, 2008

This thesis is submitted in fulfilment of the requirements of the degree of Doctor of Philosophy

Department of Electronics
& Electrical Engineering

University of Glasgow

Abstract

Existing automatic methods of analysing musical performance can generally be described as music-oriented DSP analysis. However, this merely identifies attributes, or artefacts which can be found within the performance. This information, though invaluable, is not an analysis of the performance *process*. The process of performance first involves an analysis of the score (whether from a printed sheet or from memory), and through this analysis, the performer decides *how* to perform the piece.

Thus, an analysis of the performance process requires an analysis of the performance attributes and artefacts in the context of the musical score. With this type analysis it is possible to ask profound questions such as "why or when does a performer use this technique". The work presented in this thesis provides the tools which are required to investigate these performance issues.

A new computer representation, Performance Markup Language (PML) is presented which combines the domains of the musical score, performance information and analytical structures. This representation provides the framework with which information within these domains can be cross-referenced internally, and the markup of information in external files. Most importantly, the representation defines the relationship between performance events and the corresponding objects within the score, thus facilitating analysis of performance information in the context of the score and analyses of the score.

To evaluate the correspondences between performance notes and notes within the score, the performance must be analysed using a score-performance matching algorithm. A new score-performance matching algorithm is presented in this document which is based on Dynamic Programming. In score-performance matching there are situations where dynamic programming alone is not sufficient to accurately identify correspondences. The algorithm presented here makes use of analyses of both the score and the performance to overcome the inherent shortcomings of the DP method and to improve the accuracy and robustness of DP matching in the presence of performance errors and expressive timing.

Together with the musical score and performance markup, the correspondences identified by the matching algorithm provide the minimum information required to investigate musical performance, and forms the foundation of a PML representation. The *Microtonalism* project investigated the issues surrounding the performance of microtonal music on conventional (i.e. non microtonal specific) instruments, namely voice. This included the automatic analysis of vocal

performances to extract information regarding pitch accuracy. This was possible using tools developed using the performance representation and the matching algorithm.

Contents

1 Introduction **1**

1.1 The Musical 'Object' 1

1.2 Analysing musical performance 3

2 Representation of Musical Performance **6**

2.1 Issues in representing music 7

2.1.1 Absolute/Relative 7

2.1.2 Declarative/Procedural Information 8

2.1.3 Annotation vs. Persistence of Knowledge 9

2.1.4 Structure 11

2.1.5 Mixed & Multiple Representations 13

2.2 Elements of representation 14

2.2.1 Time 14

2.2.2 Pitch 16

2.2.3 Dynamics 17

2.2.4 Musical Structure 18

2.2.5 Conclusion 19

2.3 Existing representations 20

2.3.1 Notation 20

2.3.2 Performance 24

2.3.3 Analysis 26

2.3.4 Interchange 29

2.3.5 Conclusion 31

2.4 XML for the representation of music 32

2.4.1 XSLT 32

2.4.2 Structure 33

2.5 A Specification for the representation of musical performance . . 39

2.5.1 Score 40

2.5.2 Performance 41

2.5.3 Relational links 42
2.5.4 Analysis 42
2.5.5 Deployment and Development 43
2.5.6 Summary 43
2.6 PML 45
2.6.1 PML Document 45
2.6.2 Performance 47
2.6.3 Performance Part 47
2.6.4 Event 47
2.6.5 Gesture 48
2.6.6 Cross referencing 50
2.6.7 Analytical Structures 51
2.6.8 Tuning 54
2.6.9 Separation of logical and analytical information 55
2.6.10 PML: Current implementation and Software 63
2.7 Conclusion 65

3 Score-Performance Matching **67**
3.1 What is score matching? 67
3.1.1 Types of matchers 69
3.1.2 Underlying Technologies 71
3.2 Using Dynamic Programming to Match Scores and Performances 74
3.2.1 The DP Method 74
3.2.2 Explanation of graphs and notations used in the description of correspondences 77
3.2.3 Ambiguities in the output of DTW 80
3.2.4 Cost vs Similarity 82
3.2.5 Direction of path evaluation 83
3.2.6 Efficiency 85
3.2.7 Summary 85
3.3 Polyphonic Matching using DP 86
3.3.1 Clustering 86
3.3.2 Partial Matching 87
3.3.3 Polyphonic Path Evaluation 89
3.4 Improving the DP method 92
3.4.1 The Dynamic Matcher 92
3.4.2 Interpolation 96
3.4.3 Interpolation Alignment 99
3.5 How to evaluate matching algorithms 99

3.5.1 Evaluation procedure 100
3.6 Benchmark Results 103
3.6.1 Static vs. Dynamic Matcher 103
3.6.2 Removing false positives 106
3.6.3 Score Realignment vs Static Realignment 106
3.7 Analysing real performance data 109
3.7.1 Static vs. Dynamic Matcher 110
3.7.2 Interpolating matchers 113
3.8 Conclusions and further work 118

4 Analysis of microtonal performance **123**
4.1 The Rosegarden Codicil 124
4.2 Microtonal Performance Analysis 127
4.2.1 Characteristics of performance in microtonal rehearsal . . 128
4.2.2 Vocal Segmentation algorithm 128
4.2.3 Method 130
4.2.4 Results 134
4.2.5 Conclusions 137
4.2.6 Further work 138

5 Conclusion **139**

A Performance Markup Language Annotated DTD **141**

B Vocal Segmentation Examples **149**

List of Figures

2.1 Example of bottom-up hierarchy within a top-down hierarchy . . 12
2.2 Small example of Base40 pitch notation. 17
2.3 Notation fragment generated by Lilypond. 22
2.4 Simulataneous representation of sequential and concurrent events in kern. 28
2.5 Musical example demonstrating overlapping structures 34
2.6 Example code describing musical example Fig.2.5 using overlapping tags. 35
2.7 Example code demonstrating 'milestone' elements. 36
2.8 Example code demonstrating attribute-based bottom-up hierarchy. 37
2.9 Legal method of representing multiple overlapping hierarchies within XML. 37
2.10 Example using relational links within PML to represent multiple hierarchies. 40
2.11 Block diagram describing structure of PML specification 46
2.12 Basic structure of a PML document, expressed using XML tags . 46
2.13 PML performance structure in XML 49
2.14 Scope referencing notes by IDREF list. 51
2.15 Scope referencing a range of notes using IDREFs. 51
2.16 Scope referencing a region in performance using seconds. 52
2.17 Scope referencing a region in an external resource. 52
2.18 Possible analytical hierarchy describing structure of a rondo. . . 53
2.19 Example tuning definition in PML describing 19-tone equal tempered tuning (continued in figure 2.20) 56
2.20 Example tuning definition in PML describing 19-tone equal tempered tuning continued from figure 2.19 57
2.21 Demonstrating XML namespace prefixes. 60

2.22 A musical score annotated with analytical information derived using PML. The bars display gestural information: the duration of key presses in a performance of Bach's Invention No. 1. 65

3.1 Methods of populating a grid of correspondence pairs 76
3.2 Explanation of the locations of cells as described in the rules for grid population in figure 3.3 77
3.3 Rules for traversing the optimal path through the grid. The location of cells (a,b&c) is described in figure 3.2 78
3.4 Legend describing symbol used in sequences which represent score-performance correspondences 79
3.5 Sequence describing correspondences in 3.6 & 3.7 79
3.6 DTW grid populated and evaluated using the similarity algorithms. 79
3.7 DTW grid populated and evaluated using the cost algorithms. . 80
3.8 Repeated notes causing unreliable correspondences in DP matching 80
3.9 Technical error resulting in incorrect evaluation 81
3.10 Example of ambiguous situations which cannot be resolved using DP-based matching. 82
3.11 Effect of forward and reverse path evaluation using the same rules 83
3.12 Algorithms for comparing the similarity of two note clusters . . . 88
3.13 Polyphonic DTW grid using similarity based algorithms. The grid shows that the path does not always follow the highest score. 89
3.14 Polyphonic DTW grid using cost based algorithms. The grid shows that the path does not always follow the lowest cost. . . . 90
3.15 New rules for traversing the grid using the similarity (a) and cost based (b) methods for polyphonic sequences. 90
3.16 Grid populated with Hoshishiba's cost algorithm. The optimal path does not follow the lowest cost. Correctly matching the C major triad requires the rule of first match. 91
3.17 Evaluating performance using a sliding analysis window. 94
3.18 Calculation of dynamic thresholds 98
3.19 Equation used to alter the tempo of a performance. 101
3.20 Shostakovitch Prelude No. 5, Bar 6. 104
3.21 Shostakovitch Prelude No. 5, Bar 56. 104
3.22 Comparison of the static and dynamic matchers using artificial performances. 105
3.23 Average performance of interpolation alignment with and without unalignment stage. 107

3.24 Performance of score-based alignment with and without unalignment stage at optimal alignment threshold. 108
3.25 Score-based alignment vs Static alignment 108
3.26 Score-based alignment vs Static alignment 109
3.27 Performance of static matcher on Prelude 5 110
3.28 Performance of dynamic matcher on Prelude 5 111
3.29 Performance of static matcher on Chopin excerpt. 112
3.30 Performance of dynamic matcher on Chopin excerpt. 113
3.31 Results of the static alignment matcher evaluating Shostakovitch Prelude 5 114
3.32 Results of the static alignment matcher evaluating Chopin excerpt 115
3.33 Results of the realignment matcher on Shostakovitch Prelude 5 . 116
3.34 Results of the realignment matcher on Chopin excerpt 116
3.35 Initial results of the voice independent matcher on Chopin excerpt 117
3.36 Further results of the voice independent matcher on Chopin excerpt 118
3.37 Optimal error rates for all matchers tested on Shostakovich Prelude 5 121
3.38 Optimal error rates for all matchers tested on Chopin excerpt . . 121

4.1 Real-time feedback of pitch accuracy in the Rosegarden Codicil . 125
4.2 Offline review of pitch accuracy in the Rosegarden Codicil 126
4.3 A chromatic scale in 19-tone equal tempered scale shown both in actual notation (a) and scordatura notation (b). 127
4.4 Bars 4 & 5 of *Ash* composed by Graham Hair. The second phrase of the piece is indicated. 133
4.5 Bar 6 of *Ash* composed by Graham Hair. The bracket indicates the consecutive rising chromatic steps. 134
4.6 An example of the note splitting process. The blue line represents the original note candidate. The two green lines represent the two new candidates following the splitting process. 134
4.7 Annotated score displaying the deviation from exact frequency in a performance of the soprano part of Ash (composer Prof. Graham Hair). 137

B.1 Rough (blue) & final (green) estimate of the 1st note using the pitch trajectory. 150
B.2 Rough (blue) & final (green) estimate of the 2nd note using the pitch trajectory. 150

B.3 Rough (blue) & final (green) estimate of the 3rd note using the pitch trajectory. 151
B.4 Rough (blue) & final (green) estimate of the 4th note using the pitch trajectory. 151
B.5 Rough (blue) & final (green) estimate of the 5th note using the pitch trajectory. 152
B.6 Rough (blue) & final (green) estimate of the 6th note using the pitch trajectory. 152
B.7 Rough (blue) & final (green) estimate of the 7th note using the pitch trajectory. 153
B.8 Rough (blue) & final (green) estimate of the 8th note using the pitch trajectory. 153
B.9 Rough (blue) & final (green) estimate of the 9th note using the pitch trajectory. 154
B.10 Rough (blue) & final (green) estimate of the 10th note using the pitch trajectory. 154
B.11 Rough (blue) & final (green) estimate of the 11th note using the pitch trajectory. 155
B.12 Rough (blue) & final (green) estimate of the 12th note using the pitch trajectory. 155

Chapter 1

Introduction

1.1 The Musical 'Object'

Music is manifest in many forms, such as graphical symbols on a sheet of paper, as a recording on vinyl, CD or computer, as sound waves, and as physical performance. What then is the musical 'object'? It is a misconception that music in its most tangible form exists in the vibrations that we perceive as sounds produced by the act of performance, or a recording of such a performance. A performance can never be repeated exactly and even though a performance can be recorded, listening to a recording for a second time will bea different experience The listener's memory of previous performances changes the experience of future performances. Every performance is individual to the time and environment in which it is performed. Even members within the same audience experience a performance differently due to room acoustics and personal experience. Several performances of the same piece of music may differ significantly due to the performer's interpretation of the music and yet are still recognised and considered to be the same piece of music. To take this idea further, an internet database lists over one hundred recordings of The Beatles' *Eleanor Rigby* [91]. In these recordings the music itself is wildly different, not just the performance, and yet these pieces are still recognised to be representative of the original.

Is the musical score any more tangible? Usually, when we think of a musical score we think of musical notation. Many musical notations exist, and all act as a means of preservation and *aides memoir*. *Body & Soul*, originally written in 1930 by Edward Heyman, Robert Sour, Frank Eyton and Johnny Green has been recorded numerous times. The most famous performance of this piece is

the performance by Coleman Hawkins 1939 in which Hawkins never directly asserts the melody of the original piece. So, in this case, which score is 'most' representative of the work, the score from which the original recording was performed, or an aggregate score derived from all interpretations of the original piece? Goodman believed that all interpretations of an individual piece of music comprise one complete 'work'. Therefore, every performance of a piece is representative (to varying degrees) of an overall score:

> "A score, whether or not ever used as a guide for a performance, has as a primary function the authoritative identification of a work from performance to performance."
>
> "A score must define a work, marking off performances that belong to the work from those that do not." p128 [29]

This aggregate score is not necessarily physical, but is influenced by all instances and versions of a work. In the case of classical music, the printed score will remain relatively consistent despite different editions, and performances are faithfull to the score (that is typically the intention, at least). However, in the case of Body & Soul, perhaps the only relatively consistent aspect of the piece is the chord progression, though that itself has been arranged in countless ways throughout the various 'cover' versions. In fact, *most* musical notations are 'incomplete' notations. They do not explicitly represent every note which should be played. For example, some lute tablature requires that the performer already has a knowledge of the piece, and can perform some analysis in real-time to determine notes which are not represented in the tablature.

Nonetheless, even an original score is only one representation of a piece of music. If the score was rendered straight to audio exactly as notated, the result though most probably recognisable, is very unlikely to be considered 'musical'. A score (in Common Western Notation) does not explicitly indicate how each note should be played, and in some cases does not even represent each note which should be played (e.g. ornamentation). Computer music has both blessed the world with limitless freedom to create and manipulate sound in new ways and cursed the world with the sound of scores transcribed into MIDI and played without expression, using cheap, unrealistic synthesised sounds.

A 'musical' performance is an *interpretation* of the score. The process of performance involves an analysis of the ideas and structures within the score, whether on paper, or in the mind. Based upon these analyses, various expressive techniques are used to convey emotion and highlight structures within the score to the listener.

Schoenberg is recalled as saying "Music need not be performed any more than books need be read aloud, for its logic is perfectly represented in the printed page." [58]. However, this still does not necessarily mean that the score is the musical object. As Schoenberg's idol Brahms stated, the best performance he heard was in his mind as he read the score. Therefore the act of reading a score is, in itself a performance [24]. Music is manifold, and a piece of music can be represented in many mediums. We can say that each representation is a view of the same 'object' representing different aspects of that object. However, we do not yet know what that object is. Music can be described as a process: from composition through performance and listening/analysis. Different representations can describe different stages in this process. The process itself changes with each repetition of the process as understanding of the music improves and performance techniques or traditions change.

How then do we represent something which we cannot describe, which is manifest in several different forms, not necessarily at the same time? Selfridge-Field[95] described different aspects of music as *contexts* of musical information. The phonological context represents the sound of a piece of music e.g. a recording of a performance. The graphical context describes how music should be represented visually e.g. as a printed score. The rational context which contains analytical parameters, more often termed logical information, contains a description of the *content* such as a list of notes and their properties such as pitch, duration and onset time. The gestural context describes physical movements present in the performance of a piece of music. The semantic context contains music perception and understanding. Only by representing music in multiple domains (simultaneously and synchronously) can we attempt to understand music at a deeper level.

1.2 Analysing musical performance

> "The performance of a piece of music is, therefore, the actualisation of an analytic act - even though such analysis may have been intuitive and unsystematic. For what the performer does is to make the relationships and patterns potential in the composer's score clear to the mind and ear of the experienced listener" Meyer [54](p29).

The analysis of musical performance can draw upon all of the domains mentioned above, however, the most essential information lies in the score and the performance domain. In terms of the contexts of Selfridge-Field (Section 1.1), this is the logical context of score and performance events. Performance events

may be derived from the phonological context, the gestural context, or even qualitative descriptions. However, the logical representation of this information permits analysis with respect to the score.

Whether an analysis seeks to illuminate the motives and motions of musical expression, or to explain or identify technical aspects of musical performance, the score is vital. Dixon and Widmer used a technique by Langner et al [49] developed to visualise expressive changes in tempo and dynamics called the 'Tempo Worm'. By analysing the performance in the context of the score, the use of tempo and dynamics of different performers can be analysed [84]. It was used in the analysis of particular aspects of the individual performance style of Vladimir Horowitz [112].

Palmer analysed performance errors in piano performances [103] to create a taxonomy of errors. Through an identification and characterisation of these errors, Sandler investigated aspects of musical cognition. The discovery of errors, not to mention the classification of those errors into contextual and non-contextual categories, would be impossible without a detailed description of the correspondences between the performance and the score at the note level.

A musical performance is merely one instance of a piece of music, it is subject to the time and place in which it was performed, and most of all, it is subject to the interpretation, skill and accuracy of the performer. As explained earlier, a mechanical rendition of a musical score does not yield a 'musical' performance. The reverse is also true: an automatic transcription of a performance is unlikely to yield the original score. Not only will it contain deviations from the score, intentional and otherwise, it will lack the precise notational marks which influenced the performer, thus it is impossible to tell the difference between the performer's interpretation and the composer's intention. Therefore, a transcription of the performance will not suffice as the score for performance analysis.

This document describes a new system which will permit the investigation of performance issues. The first step in the process is the representation of performance in such a way as to permit analysis. Chapter 2 describes Performance Markup Language (PML), which was developed in the absence of any other active representation which adequately represents multiple domains of musical information so as to permit performance analysis. The representation can contain score infomation and performance markup with reference to multiple sources such as audio files or gestural motion files. Extendibility is complimented by the ability to create relational links between multiple hierarchies, thus ensuring that each domain of information can be represented individually

without compromising structure by forcing information into the organisational structure of another domain of information. PML is a specification for a representation rather than a representation itself. It can be used to extend existing XML-based score representations, and the current implementation in use, which extends MusicXML, is described in 2.6.10.

A representation is pointless without tools to assist in the population and manipulation of data within the representation. Several tools have been developed to this end. Most notably, an improved score-performance matcher has been developed. Described in chapter 3, this polyphonic matcher uses score information to dynamically adjust thresholds during the matching process. Along with tools to populate PML with score and performance information, the matcher provides the basic level of information required for analysis of performance in the context of the score: performance markup, a score and event-level correspondences between score and performance.

The final chapter describes one of several ongoing projects which investigates performance issues using the PML system. This project investigates the rehearsal of microtonal music. Both real-time and post-performance feedback is used to assist the process of learning to perform microtonal music using conventional instruments (i.e. instruments not specifically designed for microtonal music such as special instruments using frets, valves or keyboards). In this particular project, the pieces are composed by Professor Graham Hair using a 19-tone, equally tempered scale. These compositions are for soprano and clarinet, thus the process of adjusting to entirely new pitches and intervals requires some accurate analytical tools.

Chapter 2

Representation of Musical Performance

Although there are several representations which describe isolated elements of performance, there is no single representation or specification which addresses performance adequately for analysis or interchange. Proposals exist for the representation of performance information such as GDIF [1] and the representation described by Hirata[35]: however the former remains unimplemented, it is focussed on the representation of physical gestures in performance, the latter represents performance information as an annotation of score information contained within MusicXML[28] which prevents adequate representation of nearly all performance issues such as structure, performance errors & multiple performances. Furthermore, it violates some of the fundamental principles of MusicXML's structure using attributes to contain data rather than metadata.

Representing musical performance exemplifies most of the difficulties which occur in the description of musical information. It requires the representation and coordination of multiple domains of musical information. The information in these domains consists a mix of both absolute & relative data and qualitative & quantitative data. Satisfying the requirements of an analytical representation which encompasses all these domains is a complex task.

The two fundamental questions which must be asked in the process of designing a representation are: "What do we need to represent?" and "How are we going to represent them?". The first sections of this chapter discusses the methodologies and design decisions which are relevant to the representation of musical information. These provide background and describe possible solutions to the two questions above. Existing representations will be discussed and

compared by their intended application. An entire section is devoted to the discussion of XML in the context of representing music.

A specification for the representation of musical information is described in section 2.5, and in section 2.6 an implementation of this specification is described which has been used in several projects some of which will be discussed in later chapters.

2.1 Issues in representing music

This section will provide a discussion on the most important issues in this area. First it will explore issues and methodologies in representation, and how they apply to the representation of musical knowledge. Secondly it will discuss the most fundamental musical elements on which a musical representation should be based. This section will discuss methodologies and representational issues which must be understood when attempting to design a musical representation. The various types of information and relations which exist in music require an understanding of these issues and methodologies and how they can assist or hinder the adequate representation of music in any domain of application.

2.1.1 Absolute/Relative

The difference between absolute and relative data is a simple analogy of the difference between the score and the performance.

A musical score is an arrangement of musical ideas, and typically the relationships between the primitive elements are all of a relative nature. The pitches, onset times and durations of notes are all expressed as relations to the properties of other items rather than static, absolute values. Only in contemporary music has there been significant use of absolute properties such as exact time or frequency in scores. Even when an exact tempo is specified at the beginning of the score, it is merely an indication of the desired tempo with more resolution than traditional textual directives.

In contrast, performance information consists entirely of absolute data. Altering the precise onset times, durations or frequencies of a performance changes the performance into something different, however, it can still be representative of the original score.

Relative information exhibits hierarchy, and thus is suited to representation within hierarchical systems. Absolute musical data has no explicit hierarchy, though it may be derived from the data through analysis. Applying musical

hierarchy to absolute performance information requires the transcription of the absolute into relative equivalences e.g. frequency into pitch.

Although humans can easily recognise the correspondence between relative and absolute quantities, algorithmically this process is non-trivial. The conversion between pitch and frequency depends on the temperament in which the performers are playing (it is widely known that, left to their own devices, musicians such as string players do not play in 12-tone equal temperament) and expressive micro-adjustments in frequency such as sharpening the leading note of a phrase. Similarly, time depends on many factors (which will be discussed in section 2.2.1)

2.1.2 Declarative/Procedural Information

Declarative information describes information about objects or events. This data must be decomposable i.e. the information in the representation must be completely independent, it should be possible to modify data without affecting the rest of the information. Procedural representations describe how to achieve a particular result. It generally consists of a list of instructions rather than a list of static information, and these systems often display substantial interactions between elements which limit data accessibility. Whereas, the information in declarative representations is far more accessible, easier to manipulate and analyse compared to procedural data.

Procedural representations are better suited to representing something which is procedural in nature. Musical performance is a perfect example of this. For example, the process required to recreate a piece for solo violin could be described as a series of instructions which indicate finger position, bow speed, direction, force & position and duration. This information is suitable for exactly reproducing a performance (though perhaps not for a human performer), or for analysing the process itself, but information such as the pitch, frequency, amplitude are at best implicit, and at worst impossible to extract without exact information regarding the properties of the violin and bow. Examples of procedural music representations include the MIDI file format [2]. MIDI time is represented as delay times between events. Therefore, the onset time of a MIDI event is not contained within the event object itself, but must be calculated from all preceding time objects.

It is common to see a mixture of procedural and declarative information within a representation. CSound [105] splits the representation of a piece of music into two parts: the orchestra file contains a procedural representation which describes how to synthesise the sound for each instrument in the piece;

the score file contains declarative data which contains the parameters for each note, and declarations of other information required by the instruments.

2.1.3 Annotation vs. Persistence of Knowledge

Where a representation is to be extensible, it becomes necessary to manage the extensions to the original system. In the case of a representation intended for analysis, there will be numerous extensions which may exist in layers of dependency. Each extension takes the representation further away from the original representation, and these extensions must be managed to ensure that the information remains accessible and compatible with existing applications. Extensions which make modifications to the original specification (as opposed to codicils) should be avoided at all costs to ensure compatibility with existing tools.

It is also possible that new information is better suited to representation in its own, native format. There is little benefit to mixing a hierarchical representation of a musical score with raw audio data in one file. The convenience of maintaining a single file is outweighed by the necessity to write new tools to manipulate the information within a new format. In this instance a multi-file representation allows information to be stored in an organised fashion, in native, cross application formats as long as the information is appropriately cross referenced. Multiple-file representations are already in use in applications such as MusicXML [28] and the Capella notation editor [77].

If new structures and information are the product of an automated process which can be reproduced, the information can be treated as non-persistent data. This data is temporary, and may be discarded once used because the results can be generated again upon demand. Information as annotation is commonplace in the internal representations of applications such as those which support musical notation. Many parameters such as precise placement of graphical objects are calculated only for display on screen. This information is represented as properties of objects within the internal representation, but the information is discarded when the music is saved in the original file format.

Alternatively, it is possible to record the algorithm used, with the appropriate parameters. The Discrete Fourier Transform (DFT) is sufficiently ubiquitous that its internal process need not be represented, and given data such as frame size, hop size and the number of analysis bins, the information can be recreated exactly. In this case the representation becomes part of a tool-based system in which the tools can be viewed as a procedural extension to the core representation.

Independent extensions which address common domains of information should, where possible be amalgamated into a single, standard extension. This permits the extensions to be included as part of the standard representation, increasing the possibilities for the exchange of tools and information.

Methods should be employed to ensure that extensions are optional and do not break the original core specification. The ideal modular system should consist of isolated components which may be included or ignored without further consequence. The GUIDO file format [38, 39] consists of three different layers of representation. The first layer includes the most basic musical concepts and the fundamental syntax of the GUIDO language. The second layer adds support for exact score formatting and more complex musical concepts, while the third level includes features which are not part of conventional music notation such as absolute timing. MEI [79] separates extended information into the analytical, gestural and visual domains. It also includes support for further independent modification of the representation through XML external entities (these will be discussed in section 2.4).

Data Abstraction

Data can often be represented in different ways, this is evident in the representation of pitch (Section 2.2.2) or time (Section 2.2.1). Data abstraction provides a transparent means of converting between different encodings. Data abstraction is performed in the application (software) domain rather than the representation itself, although it is a factor in the design of the data representation for that system.

Data abstraction is a main feature of the CHARM representation [114, 116]. The discussion of CHARM [114] refers to three analyses of *Syrinx* by Debussy. In each analysis, the encoding of pitch and time was changed. The use of libraries in the application domain allowed the same analysis program to be used in all three analyses without change to the analysis routine.

Where information can be translated without loss between different encodings, data abstraction is unnecessary. Most of the pitch encodings discussed in Section 2.2.2 are interchangeable without loss. Therefore, allowing pitch to be represented in many different ways causes more complexity in the specification of the representation, and in the application domain which must support every different encoding supported in the representation. A single encoding of pitch means only those applications which use Base-40 notation must support translation.

However, where information is lost, it is imperative that different encodings be allowed to preserve knowledge. For example, a music representation may include audio data which could be encoded in many different formats. The representation of time is different in each format, and the format it uses contains information regarding the content e.g. bandwidth can be derived from the frame rate. If pitch information is represented in MIDI integer representation it is wrong to translate this automatically and irreversibly into a more complete representation of pitch because the ambiguity of pitch spelling is lost, arbitrary data is added, and false information may be taken as truth.

2.1.4 Structure

There are many different structural forms for representing data. Generally, on the macro scale, a representation will mostly conform to one structural form (e.g. top-down hierarchical representation) however, for complex representations it is possible to find a mixture of structural relations.

In an explicit structure the structural relations of the objects determines the organisation of entities within a representation. In fact, some structural items can be considered entities themselves. This is particularly the case in XML-based music representations in which notes are often contained within a hierarchy of *Score* → *Staff* → *Bar*. Implicit structure is derivable from explicit data within the representation, for example, MIDI notes within a standard MIDI file are unstructured, however, structure can be extracted using the channel information associated with each note.

A tacit structure is one where structural information is absent. The knowledge is within the data, but it is inaccessible (perhaps limited by current technology). For example, most audio files exist as streams of values with no representation of structure. It is possible to extract structure from the information, but it is not accessible

Hierarchical relations are those which define an element as *part-of* another element, or declare that an element contains another element or elements. Top-down hierarchies, which use the *contains* relation are common in representation because they organise information in an accessible and intuitive manner. Bottom-up hierarchies are also common, and can be used to represent hierarchy implicitly within the data of some other structural scheme (which may exist within another hierarchy). The example in figure 2.1 demonstrates the use of a bottom-up relationship to implicitly represent a micro hierarchy within a top-down macro hierarchy using element attributes. This mixed hierarchy can be seen in MXML and MEI to indicate structures such as beams. The methods of

representing his sort of relationship in XML representations such as these will be discussed in section 2.4.

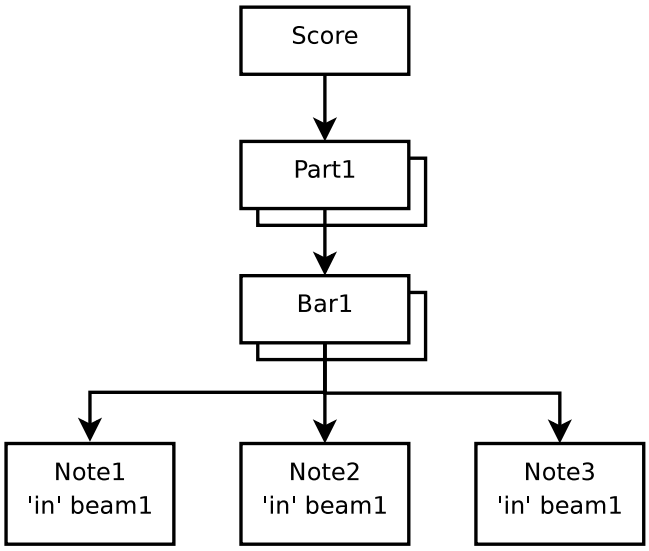


Figure 2.1: Example of bottom-up hierarchy within a top-down hierarchy

Relational representation identifies referential links from one object to another. These links may be one-way or bidirectional. These links are part of the objects, and do not explicitly define structure, though they can be used as in figure 2.1 to indicate implicit structure within a mixed hierarchy. Relational links usually refer to other objects using unique identifiers. SGML and XML provide support for this type of link through their use of ID and IDREF attributes. An ID attribute must contain a value which is unique among all other ID attributes in the document, while an IDREF must contain the ID of a existing element within the representation. This is used most recently in SMDL[96], MusicXML[28] and MEI[79], but also exists in older representations such as Charm[114]. It is also the basis for relational databases in which objects are given unique keys. Structural relations are defined by use of these unique keys. In a relational model, the order of data objects has no bearing on the content (though the data may be ordered for the purposes of optimising performance). These references can be used to create bi-directional, or many-to-many relationships.

2.1.5 Mixed & Multiple Representations

Mixed representations use a variety of representational schemes to encode different parts of a document. In multi-domain representations it is common to represent data in different domains in the scheme which is most suitable. This depends on the type of data, and the type of access required. Multiple representations contain multiple *views*, or *contexts* of the same object. Multiple representational schemes require careful management to ensure consistency between different contexts

The different musical contexts as described by Selfridge-Field [95] can be thought of as different perspectives of the same object. Each perspective contains different types of raw data and different analytical structures. Therefore an adequate codification of music in more than one context must be a manifold representation. Each context will contain different information with different structural demands. Therefore it will also have to use mixed representational schemes to ensure that each domain of information is structured in a way which makes the data within as accessible as possible. All of these domains must be interlinked and the consistency of information between contexts must be maintained. This is by no means a trivial task. The resulting system is likely to consist of many different sub-representations (e.g. support for the many different representations for audio data alone) and many different tools for retrieval and analysis. SMDL and MEI have both been designed to accommodate multiple contexts of information. MEI includes support for the analytical, gestural and visual domains within its representation. Unfortunately, these domains are implemented as optional attributes of elements in the standard hierarchy, therefore the ability to represent each domain is limited. As an example, the gestural domain contains additional note attributes such as MIDI pitch and onset time which combines the logical/graphical score domains with the gestural domain. This allows the performed attributes to disagree with the notated attributes (indicating a departure from the written score), but does not offer an adequate solution to the representation of performed notes which are not notated, or notated notes which are not performed. The lack of structural independence means that this representation is neither truly mixed nor multiple. The SMDL specification splits the same domains as defined in MEI into isolated representational schemes which can be interlinked using various methods which were developed in the HyTime specification[26]. While this representation could be used to represent multiple mixed representations of a musical object, the logical score was the only domain described in the specification, and SMDL was never adopted as a standard representation.

2.2 Elements of representation

This section will discuss the basic elements of music and how to represent this information within a computer representation. The three qualities of time, pitch and dynamics are considered the most fundamental properties of a note. The musical structure in which notes occur binds these primitives together within a piece. All of these elements have been represented in many different ways and a discussion of these methods is necessary to comprehend how to represent this information optimally for any given application.

2.2.1 Time

The issue of representing time can be split into three discussions: the representation of relative time, absolute time, and, if necessary the coordination of these two domains.

Relative musical time is most easily represented by an arbitrary number of divisions. Generally the number of divisions is the highest common divisor of every onset and duration which must be represented. In a polyrhythmic piece such as Chopin's Fantasie Impromptu opus 66, the highest common divisor splits the bar into 672 divisions. This is certainly not convenient for human analysis, and in many MIDI related sequencer applications, the concept of divisions is used in a more user-friendly way denoting a relative time as a combination of bars, beats and subdivisions. This has similarities with SMPTE time [64], except the content determines the resolution rather than the hard/software. Relative time is also often described using bar, beat & subdivisions. This notation can be derived from the total number of divisions by analysing all the time signatures prior to that time. However, in polymetric pieces, a particular number of divisions may relate to different measures depending on the staff.

Alternatively, time could be expressed as integer ratios of the measure or beat. As an example, Lilypond represents the timing of triplets as follows:

```
\times 2/3 \{c4, e4, g4\}}
```

This indicates that the duration of two notes is occupied by three notes, therefore the notes have a duration of two thirds the notated value (a crotchet). This is easily legible from a human perspective, which is essential for a human-entry representation such as Lilypond's [59]. It is also independent of any other properties such as a global resolution, but can increase complexity in applications.

Many notation representations only represent temporal position implicitly (MusicXML[28], MEI[79], Wedelmusic[109] to name just a few). Musical notation does not necessarily require a value for the position of each note within time, the order of notes and information regarding the voice or layer is enough to typeset music graphically. The process of deriving an explicit start time, given sufficient information is not difficult, however in the case of a representation which is intended to be used for analysis, it is important that the properties of an object can be retrieved easily, which requires that information be explicit and declarative.

Several systems exist for the representation of absolute time. SMPTE [64] was developed by the Society of Motion Picture and Television Engineers to represent the time of individual frames for film, video or audio material. Time is encoded as a binary representation of 'hour:minute:second:frame' and is recorded straight onto the film or video on which the material is recorded allowing material to be synchronised during editing or playback. SMPTE timecode is also used to synchronise equipment in a studio setting, and MIDI Time Code uses a version of SMPTE time code which is transmitted across MIDI cable to synchronise MIDI devices. Many file formats which contain streams of data, represent time as an external entity (i.e where the time is not an internal property of the event itself). Formats based on IFF or RIFF (discussed further in 2.3.2) stipulate a standard frame rate and define the amount of data contained within each frame, the time and track is thus calculated from the position within the file.

One might be tempted to consolidate the representation of different streams of information with different frame rates into one timeline. Although representing time as seconds would seem convenient, the digital representation of floating point numbers will introduce rounding errors which can cause inconsistencies in the conversion between floating point times and frame numbers. Indeed, the frame rate contains information about the signal itself. The framerate of audio signals indicates the bandwidth of the signal, and the framerate of SMPTE is an indication of the video encoding used such as PAL or NTSC.

Representing the mapping between absolute and relative is a complex task. It has been shown that a simple tempo curve is a naive model of musical time [19]. Objects which occur simultaneously in the score, such as chords will appear sequentially, and in any order within the performance. Different parts, even within the same instrument (for example separate hands in a piano performance) may perform at different tempos within a phrase such that notes in a performance occur in a separate order than they appear in the score. Marsden

extensively explores the representation of musical time and described a formal logical approach to its representation [52]. Honing has also developed a calculus for representing musical expression which attempts to describe and permit transformations of expressive timing [20].

2.2.2 Pitch

The most ubiquitous representation of pitch does not actually represent pitch at all. Rather, MIDI [2] assigns 128 discrete frequencies, and these frequencies are selected by a number in the range 0-127. The frequencies are, by default, tuned to a 12-tone equal tempered (12ET) scale at concert pitch. MIDI was originally designed as a protocol for real-time communication between musical devices, and for this purpose the numbering system is adequate. Music which uses tunings other than 12ET are now supported using the MIDI Tuning Standard (which is now part of the standard MIDI specification) although the number of discrete frequencies is limiting for scales which divide the octave into more than twelve intervals. Unfortunately, the vast majority of musical applications which use MIDI have adopted this numbering system as a 'representation' of pitch. This is partly because applications do not support notation, or because the vast majority of users no longer use notation as a means of composing or because applications wish to represent information in a format which is as close as possible to the format it will use in real-time. Although an integer system of pitch representation such as MIDI cannot distinguish between different spellings, this representation is useful in the context of comparing notes for enharmonic equivalence.

MIDI aside, the most common way to represent pitch is as a combination of note class, octave number and accidental. This is also often used as a human accessible way to represent MIDI pitch. While this representation adequately represents enharmonic spelling, there is considerable difference in the numbering system for octaves (middle-C is variously denoted by numbers in the range 3-5). This is termed the *note-octave* or *pitch class-octave* system of pitch representation.

Graphical applications such as those which transcribe images of sheet music often represent pitch using a combination of clef, key signature, accidental and height on staff. This methodology represents position of graphical objects rather than pitch *per se*, although the pitch is implicit in the combined attributes of the graphical objects.

These representations offer a simple and accessible method of representing musical pitch. However, many processes such as diatonic transposition or in-

terval analysis are extremely difficult, if not impossible under these schemes. The interval between $C\#4$ and $B\flat4$ is a diminished seventh, and the interval between $D\flat4$ and $B\flat4$ is a major sixth. Although in 12ET music the interval size is the same, the two intervals have distinct functions in the context of harmonic analysis. Other pitch representations use a form of integer representation which allows these complex musical transformations to be calculated by simple arithmetic. The Base40 representation [33] represents individual pitch spellings as integers $C\flat\flat1 = 1$, $B\#\#1 = 40$ (not all integers correspond to an actual pitch). Similarly, intervals are assigned a delta value which corresponds to the numerical distance between two notes which are separated by the given interval.

Pitch	Base40 Integer	Interval	Base40 Interval
$C\#$	4	Dim. 2nd	4
$D\flat$	8	Maj. 6th	29
$B\flat$	37	Dim. 7th	33

Figure 2.2: Small example of Base40 pitch notation.

Binomial representations [9] use integer pairs to represent 'pitch class' (pc) and 'pitch name' (pn). Pitches which share a staff position, or 'letter name' belong to the same name class e.g. $C\flat$, C, $C\#$. A pitch class defines all of the enharmonically equivalent pitch spellings e.g. $B\#\#$, $C\#$, $D\flat$. Operations on these numbers allow transposition between pitches and a comparison for enharmonic equivalence.

These representations are less accessible for applications whose focus is the performance of music, or the graphical display of notation. However, with the exception of MIDI pitch, a pitch within any of these systems can be translated into any other system with a simple lookup table. Some, MIDI-based sequencer applications e.g. Rosegarden [81] which support western musical notation augment MIDI pitch with an (optional) accidental. The inclusion of an accidental, if mandatory for all pitches, provides enough information to translate between the other pitch representations.

2.2.3 Dynamics

Dynamics in the context of a score may relate to a single event, a group of events or a range in relative time. Graphical representations may or may not make an association between a dynamic score marking and the events to which it relates. The performance of these directives is very much dictated by individual

interpretation.

The absolute measurement of dynamics is typically measured in either dB. MIDI has only has the ability to represent volume on a per-channel basis. In many cases the gestural quantity key velocity, which is associated with each MIDI event is used as an indication of dynamics. Again, this is limited to the range 0-127, and there is no standard relationship between velocity and dynamics. In fact some MIDI instrument have various mappings between physical velocity and MIDI velocity. Therefore it is impossible accurately to capture dynamic information or to produce precise dynamic output without a detailed specification of the recording or synthesis device.

2.2.4 Musical Structure

The western musical score can be (partially) described as organising musical elements in two dimensions. The vertical axis organises the information by pitch and, in the case of multiple staves by instrument also (though the score is by no means a pitch/time graph). The horizontal axis organises information by time. Further structural organisation can be seen as superimposed upon this basic organisation. This allows for immediate grouping of information with respect to either time (events in a vertical region) or staff (all events on the same horizontal region or staff). Few representations are capable of imitating this duality, perhaps partly because of the nature of digital files, in which information natively stored in streams, and partly because representational structures are imposed upon them by the choice of representational language. Representational languages such as XML and SGML impose a hierarchical structure to data-based information. In these cases, it becomes necessary to address the issue of two dimensions by using parallel and sequential entities. This problem is most clearly demonstrated in MEI [79] and MusicXML [28]. Each of these representations exist in two versions which contain exactly the same information. The difference between the two variations is that one groups notes according to staff, while the other groups information according to time. In a *bar* $\rightarrow$ *staff* representation, the higher structure is the bar element which groups all events which occur within that bar. Objects within a bar are further grouped according to the staff to which they belong. Information in a *staff* $\rightarrow$ *bar* representation is first arranged by staff, and each part contains a sequence of bar elements in which notes and other information are stored. Because the information in each variation differs only in organisation and not in content, it is possible to convert between the two schemes. This conversion is further simplified by the use of XSLT 2.4.1.

A *bar* → *staff* representation or time-wise structure is slightly more suited to analysing harmony because information regarding all score objects within a time region are accessible together. It is also more convenient for coordinating orchestral scores. In a *bar* → *staff* representation or part-wise structure the analysis of melody and individual parts more accessible. Although each method is equivalent and interchangeable, retrieving information is quicker and easier using the appropriate structural scheme.

However, at the note level, the *bar* → *staff* representation still represents music as individual sequences running in parallel, therefore harmonic information is only slightly more accessible than in the *staff* → *bar* organisation. For example, multiple voices in MusicXML may be represented within a single staff-bar. The convention within MusicXML is to represent multiple voices sequentially, using a *backup* instruction to indicate that the temporal position of the following notes does not continue from the preceding element. Although simultaneous notes which appear in different voices can be stored contiguously, it results in an extremely complex, and confusing non-linear representation of time. Furthermore, in polymetric pieces, it is impossible to arrange music into a coherent *bar* → *staff* representation because the bars are not equal across all staffs.

Kern [44] represents elements within a two dimensional space similar to common western notation. The 2D space is contained within an ASCII file. Simultaneous events are arranged in rows and sequential events are arranged in columns. Therefore all the events within a particular part occupy a single column of data, and all events which occur at the same time appear on the same horizontal line. Whereas the *bar* → *staff* hierarchy only partially supports harmonic analysis, the kern system allows simultaneous analysis of musical information as parallel or sequential data within one organisation.

2.2.5 Conclusion

Representing musical performance for analysis must cater for many different tasks. The information must be displayed graphically, it must be accessible for analysis, and it must also synchronise relative and absolute information. Therefore in most situations it is advantageous to use the most versatile system in representing different elements of music.

In light of this, the most suitable representation of pitch is the note-octave class. This is the easiest system for human comprehension and graphical representation, and is also the system most commonly used in opther representations. In any case, analytical applications which would require calculations which can be evaluated trivially in numerical systems such as Base-40 can convert 'on-the-

fly' from the note-octave representation into whichever system best suits the analysis.

Relative time based on divisions is far more suitable for analytical applications. Divisions allows the position of notes in time to be represented declaratively. It allows simple synchronisation within polymetric music. Furthermore, using bar-beat-divisions aids graphical and analytical applications while maintaining a system based on divisions.

A performance analysis representation may have to reference various sources of performance information with different representations of real time. Therefore it should either be capable of representing time in multiple formats, or should represent time in a neutral format such as seconds, ensuring that it may be recorded with adequate precision to represent individual frames within external sources.

Common Western Notation is a difficult representation to aspire to

2.3 Existing representations

There are countless different computer representations of music. A comprehensive representation of music, in any domain other than the phonological domain, has yet to emerge. The variety of information forces the design of musical representations to be an exercise in feature selection. Representations either cater for one specific application or musical style with precision, or offer offer a broad range of features which will suffice for mainstream popular or classical music, "a Jack-of-all-trades, master of none". This section will examine some of the most current and relevant representations in the context of the application domain for which they were designed, allowing a discourse for comparison between those which broadly aim towards the same goal. The representation presented here is not intended to be

2.3.1 Notation

The most obvious application for a musical code is in the creation and archiving of musical scores. Musical codes which deal exclusively with notation can be divided into three groups: codes which are designed for user entry, archival and interchange of logical score and archival and interchange of absolute graphical information.

Typesetting

Since graphical user interfaces have become more advanced, text entry has become a generally unpopular method of generating musical input for computers. However, text entry codes continue to be used and developed, and, after a steep learning curve, users report considerably faster data entry than using the now conventional 'point & click' method. Examples of text entry codes include abc[107], Plaine and Easie[41], Common Music Notation (CMN)[89] and Lilypond[59]. The main objective of such a representation is to represent musical notation in a plain text format which is easy to read and easy to edit by hand. Usually the code is processed through a typesetting algorithm to produce graphical output, although it is also possible to generate MIDI files from some formats.

Lilypond is the most advanced and widespread user entry code for musical typesetting. The Lilypond syntax has been constantly updated to improve the input system, something which is achieved at the expense of backwards compatibility. Pitch in Lilypond is represented using a form of note-octave notation where the octave is expressed as an offset relative from a reference octave. Beams and slurs/ties are expressed by enclosing the affected notes in braces or parentheses. The typesetting algorithm, and graphical font have been based on hand-engraved scores which produces very attractive output. However, Lilypond allows very exact layout constraints to be specified should the score require extraordinary notation.

abc is a far simpler language than Lilypond. It is particularly popular amongst folk music communities, and a huge corpus of work is available freely on the internet. abc was originally developed to simplify the creation of input files for another language: MusicTex. MusicTex is a musical typesetting tool based on the TeX processing utilities. It suffered from an extremely verbose dialect, and the abc2mtex software provided the ability to create score typeset by MusicTex using a far simpler input language.

The syntax used by Lilypond and many other text entry systems use a free markup representation where different structures may overlap for example, the musical fragment in Figure 2.3 contains two overlapping structures: a beaming structure and a slur. This is represented in Lilypond syntax as:

```
c[ c(] d)
```

This form of representation is easily parsed, and easy read and edit, however information such as time is usually implicit, and without a strict hierarchical nature, membership of hierarchies also becomes implicit, therefore information

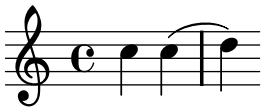


Figure 2.3: Notation fragment generated by Lilypond.

thus less accessible for processes which require random access such as analytical algorithms.

Logical

Storage, interchange and analytical applications typically represent a musical score in the logical domain. The accessibility of information in the logical domain make this domain essential when information must be encoded in an application neutral (in terms of software application) form.

MuseData is an ASCII format which aims to represent a musical score in a software neutral fashion. It consists of two 'layers', the first comprising a very limited representation which consists of pitch and duration information only. The second layer may comprise other information such as information for printing scores or MIDI data. The MuseData archive contains many scores which can be converted into different formats such as MIDI, DARMS[93] or kern[44]. The 'MuseData Universe'[94] is a system which includes the MuseData archive, the HumDrum syntax and the HumDrum toolkit. The archive acts as a source of musical scores which can be converted into representations based upon the HumDrum syntax e.g. kern (see below) and analysed using a set of tools contained in or based on the HumDrum Toolkit. MuseData has the capability to be extended by extending the number of attributes of a note. However, the basic level is a very sparse representation of a score, and MuseData is not a convenient representation for describing analytical structures.

DARMS (Digital Alternate Representation of Musical Scores)[93] was developed to provide a means for writing music with an ordinary keyboard. It later grew to become a widely used representation for analysis and production of commercial scholarly editions of music. Several different dialects exist and, although all can be reduced to the basic form Canonical DARMS, there is little compatibility between different systems. DARMS is a concise format, but it is not easily human-readable: durations are represented by a shorthand alphabet, and pitch is specified using staff position, where the first line of a staff is 1 plus some multiple of 50.

Absolute

The final class of notation representations encode a score as musical symbols which contain positional and dimensional information. Graphical files are unsuitable for any application other than the production of scores, or the analysis of issues relating to typesetting. Fundamental musical attributes such as pitch and duration are not explicitly expressed, and thus conversion to more generally useful codes is necessary to use data in other application domains.

NIFF (Notation Interchange File Format)[15] is a binary format based on the RIFF specification. Notes are defined by note head, duration (expressed as a rational number) and position relative to a staff. NIFF[15] has also been implemented in XML (NIFFML) [13]. NIFFML does not alter the structure or content of NIFF: it is a strict implementation of NIFF in XML. NIFF is a common interchange format for optical recognition software which scan musical notation.

Strangely, there is only one utility which converts from a user entry format to a format which represents absolute graphical information. The Score [98] programs use two separate codes: a logical input code which is an human readable ASCII encoding of logical score information and a parametric file which includes graphical objects and exact positioning. The Score program uses the input file to create the graphical file, which is still able to be imported using commercial notation packages.

Composition

Representations which are intended to be used as a compositional tool (excluding those which represent the Common Western Notation) are typically used within the electro-acoustic genre. These representations often have a procedural, even object-oriented[70] aspect to their representation, which allows processes to be applied to defined and applied to multiple groups of objects.

The CSound representation consists of two files, a score file and an orchestra file. The orchestra file defines the synthetic instruments in a language somewhat similar to the C programming language. The score file contains a declarative list of note start times, durations and other parameters which are used to create scheduled input for the instruments.

Canon [18] was designed to provide a means of generating note level control information for synthesisers. It was built upon the declarative programming language LISP, and the Canon score, like other representations such as FORMES[82], or Smoke[70], can be viewed as 'programs' [18] which contain, in one form or another, a set of procedures and a set of parameters which are

used when those procedures are activated. Although it is possible to envision the score as a list of parameters and processes to apply, the underlying parameters (notes and their properties) are far more accessible. In object-oriented or procedural languages, these properties are either implicit, or absent unless the 'program' is evaluated.

2.3.2 Performance

Audio

A performance is one instance of a piece of music, and performance data is unique to that performance. The most ubiquitous, and in many ways the most comprehensive representation of a musical performance is an audio recording of the performance itself. Various formats exist for the encoding of audio data. Most of them are proprietary formats which exhibit no significant difference in the encoding of audio information whatsoever. For example the Interchange File Format [92](IFF) specification (on which Apple's AIFF format is based), and RIFF used by Microsoft's WAV format differ only in their use of big-endian or little-endian numbers. These formats typically represent audio information as a stream of amplitude values coded as up to 64-bit integers or floats at a fixed framerate with interleaved channels.

Audio information arguably contains the most significant information compared to other domains of performance information. However, information in the audio domain is also among the most difficult to extract. Despite decades of effort, automatic transcription, score-following and even beat-tracking are still very active fields of research, and the problems within are far from solved.

MIDI

The availability and convenience of various MIDI devices has led to common use of MIDI as a representation for performance information. However, there are several fatal flaws in the use of MIDI as a representation of a performance. These flaws are partly to do with the limitations of the protocol and file format themselves, but also to do with the way the data is treated. MIDI represents notes as an event pair which denote start & end time, key number (pitch) and beginning and end 'velocities'. A MIDI device is limited to 128 frequencies on a single channel. Although the MIDI Tuning standard was introduced to allow the frequencies of notes to be changed, few synthesisers or sequencers support this feature. It is also unclear whether synthesisers should change the frequency of a note which is playing when a new frequency is transmitted, or apply changes

only to new notes. Otherwise, the only way to alter the frequencies of notes is to use pitch bend which alters the frequencies of all active notes on the channel (the range of pitch alteration may be changed, the default is ± one tone).

Not only is the control resolution insufficient for detailed analysis, but value ranges have no set reference point. Many MIDI controllers offer several different velocity curves which map the velocity of the physical gesture to the standard MIDI range. Thus, it is impossible using only the MIDI information to obtain reliable, consistent data. Beyond the limitations of the representation itself, the information recorded from MIDI instruments is often treated as the wrong type of information.

MIDI information is *gestural* information. The *NoteOn* and *NoteOff* signals do not signify the beginning and end of a note. They represent the beginning and end of a gesture. It is impossible to determine the duration of notes recorded from a piano performance using MIDI for two distinct reasons. High notes have a very short audible duration, thus, there is no guarantee that a note is still audible, purely because the key is still depressed. Secondly, When the key is released, the note will continue for an indeterminate amount of time if the sustain pedal is depressed. For further discussion on the inadequacies of MIDI, see Moore [55]. Efforts have been made to compensate for some of these inadequacies by synthetically modelling decay rates [21], however, accurate data cannot be derived without an analysis of other data mediums.

Unfortunately, because MIDI is adequate for the majority of commercial music and most home users, MIDI devices remain the industry standard, and therefore, in some circumstances, the only source of viable gestural information.

Other formats

Most formats for the description of gestural information are proprietary formats used by the manufacturers of gesture tracking systems. These representations typically consist of two parts: a skeletal description and raw gestural data. The skeletal description describes the physical relationship between objects which are to be tracked. The raw data contains sampled data which describes the movement of each of these objects either in real-time. The BVH [5] (BioVision Hierarchical data format) is the most common proprietary gesture format. This plain ASCII format contains a hierarchical description of the skeleton of the object. This skeleton is described as a hierarchy of joints, with terminating 'end points'. Each node is accompanied by a description of each channel of data which describes the movement of the joint: position or rotation in each of the three dimensions, and also the initial state of the joint (as an offset from its

parent joint) before tracking commences. This also defines the order of elements within a frame of motion data.

The C3D format[100] is intended to be an interchange format for gestural data. Developed in the context of biophysical science, C3D contains extra metadata describing the human subject e.g. age, height, weight, date of examination etc. Data encoded in proprietary formats is intended to be translated into C3D format to allow a single set of tools to be used for all data.

GMS (Gesture and Motion Signal) [25] is a low-level, generic format for the description of gestural data. It is a binary file based on the IFF file specification. The lowest level of information is contained within tracks. This may be positional or acceleration information in one dimension. Several tracks can be grouped into one channel, this information would combine to describe the complete motion data of one point. Units provide structural dimensionality of gestures. A unit should contain channels which are not dynamically independent, thus a unit may contain all the channels of a hand, or all the channels of a human body. Finally, units are contained within scenes. The framerate is defined in the scene header section and is thus constant and consistent within each scene. GMS offers no representation of analytical structure, and does not even describe the content of information within a track/channel. Each hierarchical component within GMS can be named, which could be used to determine data content as long as naming conventions are created and maintained. For example, the track name may be used to define the type of data such as position in X plane, acceleration in Y plane.

GDIF (Gestural Description Interchange Format)[1] is a new proposal for the representation of gestural information. It is not, as its name might suggest, derived from the IFF specification, though it is inspired by the GDIFF format, which is an IFF format. GDIF aims to combine the representation of all elements of gestural description such as raw data, segmentation, semantic descriptions. It is also hoped that the representation of gestural primitives such as trajectory, force and pattern can be formalised. Qualitative descriptions of movement will be supported by the use of Laban Movement Analysis (LMA). Unfortunately, this representation remains a proposal, the only implementation which exists is a map of OSC address spaces which are used to capture data in real time.

2.3.3 Analysis

Although a number of analytical music representations have been proposed, only one representation (Humdrum) has succeeded in becoming the core representation of a substantial number of musicological investigations. Harris et al [115]

proposed that analytical representations should be judged on two criteria: *expressive completeness* and *structural generality*. Expressive completeness refers to the range of raw musical data which can be represented. Structural generality refers to the range of high-level structures which can be represented. In their evaluation of current representation of the time, the waveform of a recording of the music determines the upper-bound for expressive completeness while offering no structural generality. Structural generality is achieved through extensibility and adequate support for the description of relations between elements. However, analytical representation must also make data accessible, which generally requires explicit declarative information. The information contained within Smoke, which scores highly in terms of structural generality, is less accessible than that of other representations which have less structural generality.

Humdrum has been used in analyses as diverse as examining scale systems in Korean court music [56] to investigating symmetries in notated musical dynamics [43]. The actual representation of the Humdrum toolkit is not fixed, and is intended to be tailored for the particular task in hand. Data can be imported from other sources including a database of musical scores encoded in kern (the 'vanilla' Humdrum encoding). Kern represents only the most fundamental information contained in western musical scores. It is an ASCII based syntax which defines a specification for the representation of musical information which represents both sequential and concurrent relations simultaneously. Concurrent events are represented on the same line, while sequential events are represented in columns. Figure 2.4 contains a simplified representation of the first two bars of Shostakovich's Prelude 5. This method was used previously by Brinkman [10] in an internal computer representation of music. Brinkman represented musical elements as nodes within a two dimensional, doubly-linked list. Each element within this matrix contained links to the next and previous element both in sequential terms and harmonic terms. In the same paper, Brinkman used the representation to perform a simple harmonic analysis of polyphonic music. Brinkman, however, was unable to find an acceptable way of translating this internal representation into one which was suitable for external storage. Interestingly, Brinkman's external representation uses an ASCII line/column representation which contains different attributes in separate columns, but fails to extend the lines to include concurrent information across all parts. Instead, the music is organised in a time-wise fashion with elements in temporal order grouped by part within each bar. In a similar example, Prather [71]uses the same two dimensional doubly-linked lists to represent music internally, but relies on SML [72] to represent the data externally.

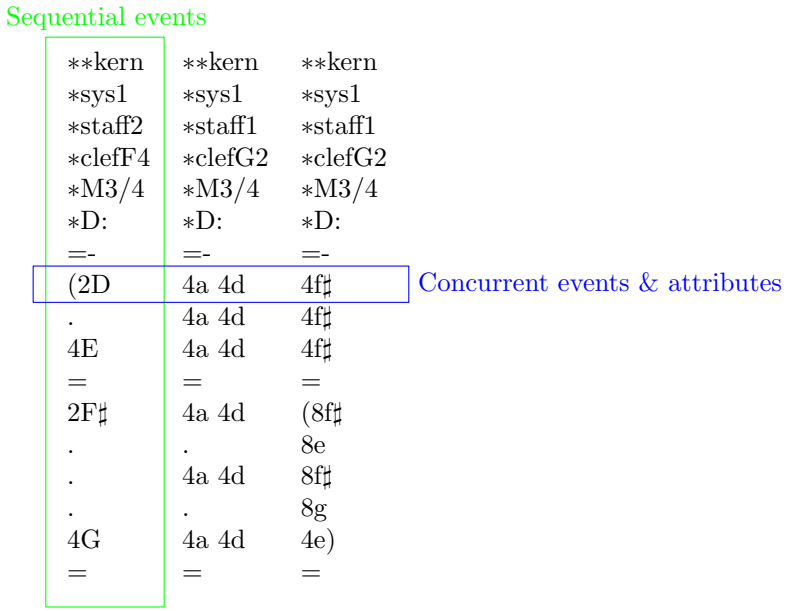


Figure 2.4: Simulataneous representation of sequential and concurrent events in kern.

Although Humdrum is a successful implementation of the bi-dimensionality of basic musical structure, and has proven to be a popular choice among musicologists, it remains, like its two predecessors, a temporary format. In the 'MuseData Universe' [34, 94] musical scores are stored in either MuseData or kern format (which is an inadequate representation of the musical score). There is no persistence of analytical data, therefore if further analyses rely on the data of previous analyses, the data must be generated again, every time. However, Humdrum does specify a number of standard extensions called 'schemes', which allow the representation of particular information (e.g. Schenkerian graphs). This allows multiple implementations of a particular analysis to be used interchangeably.

CHARM is another specification for the representation of musical data, rather than a representation itself. It is designed to be application independent, though the only evidence of its use is in the context of analysis, thus it will be discussed here. CHARM [116, 114] is a 'general musical representation system which is not based on any particular music style, tonal system, tradition or application'. CHARM uses abstract data types, which means that it should be possible to represent information such as pitch in any encoding. An access library provides a layer of abstraction which will convert between the stored encoding and the encoding desired by the application [117].

CHARM achieves structural generality through the use of relational links and structural constituent elements. Constituent elements can contain note elements, or other constituent elements. All elements have a unique identifier which is used to reference that element. Thus CHARM is capable of denoting multiple hierarchies using ID-based relations.

The Sound Description Interchange Format SDIF [118] is a format which allows the representation of analytical representations of audio. It is based on the IFF standard, and specifies a standard representation for time domain representations, short term Fourier transforms and fundamental frequency estimates.

Sonic Visualiser[11] is an audio analysis application. It allows the extraction and description of information from audio files. It uses the VAMP plugin architecture which defines a standard API for plugins which process audio and return symbolic information. Sonic Visualiser saves sessions in a compressed XML file which contains the markup information, and the audio file with which it is associated. The audio file itself is only referenced in the session file, the actual audio data remains encoded in the original file format.

2.3.4 Interchange

In the absence of any standard representation of music, interchange codes form a stepping stone in the conversion from different formats. The advantage of interchange formats is that developers need only create the tools to translate between their format one other format: the interchange format to support the exchange of data between many different formats. The disadvantage of interchange formats is that there is no guarantee that certain information will be present when importing information from an interchange format, or that information will be preserved when exporting to other file formats. Interchange formats require a large set of features which must be developed carefully to ensure interoperability between the many different formats they support.

Until recently, MIDI was the dominant interchange format between applications such as sequencers and notation packages. Musical scores were encoded as MIDI note events using relative time onsets and durations, this, to some degree, preserved temporal information, though pitch information and score markings were lost. Recently, MusicXML [28] has achieved widespread support both in the commercial and open-source community. It is an XML-based representation which permits the exchange of musical scores.

MEI [79] is another XML based representation of the musical score. It bears many resemblances to MusicXML in general structure. MEI however, has never achieved any use as an interchange format since there are no utilities to convert

to or from MEI to any other format [78].

The NIFF (Notation Interchange File Format) was designed to facilitate conversion of graphical encodings. It is now supported by several optical score recognition applications, and an XML implementation has been developed.

Multiple Application

There are representations which are intended to be application neutral. The Standard Music Description Language (SMDL) [96] was based on SGML and was developed in parallel with the HyTime specification [26]. SMDL was developed to represent musical information in 4 domains: Logical, Graphical, Analytical and Gestural. HyTime [26], in the context of SMDL, provided the means to coordinate and correlate the various domains. It allowed mappings between different representations of time, and allowed relations to be described both within the document and using external sources. The SMDL specification states the main purpose as the interchange of musical information. However, it has also been suggested that SMDL was an 'enabling standard' [97] from which only the necessary modules need be used. However, the SMDL specification only detailed the logical representation of musical data (and through HyTime, object relations and temporal information). It seems that SMDL was lacking in the description of many areas, but the areas which it did describe were considered too complex [14]. To extract a small portion of the SMDL architecture which was relevant would require an in-depth understanding of both SMDL and HyTime. However, SMDL never gained the support of the community and the working group was officially closed on 23$^{\text{rd}}$ May 2006 [36]. The only known application of SMDL was an attempt to use SMDL as the basis for a database of musical information.

The MuTaTeD project combined SMDL with the graphical representation NIFF. A graphical compiler was used to generate NIFF from logical SMDL files.

For a representation to succeed, it must be supported by applications. Unless people are willing to create the tools required to create, manipulate and use the data contained within, it will be impossible to use the representation. This was perhaps the ultimate reason for SMDL's demise [97] despite the backing of international committees, and is a clear warning to the developers of new representations and standards.

Even now, with the benefit of hindsight, there is a lack of available tools for the display and manipulation of scores. Commercial notation applications allow the creation and manipulation of scores, but their proprietary nature prevents their use in novel applications such as clients for database query applications or

the analysis of musical information. Until an open standard API for the graphical representation of musical scores exists, the development of software and thus the development of representations and research projects will be severely limited.

2.3.5 Conclusion

Current representations of performance are adequate for the retrieval of performance information only, not for the analysis of performance issues, which requires the coordination of several different domains of information. Although no adequate representation exists, lessons can be learned from existing representations which successfully represent information for different applications in order to derive a specification for such a system.

Although existing representations of performance, by themselves, are inadequate for analysis of performance issues, it is necessary to use these representations in such a system. Audio representations are essential to retrieve information about the performance. Even MIDI, despite its various shortcomings, provides a convenient method to record performance events, as long as this information is encoded in a more suitable representation scheme. A suitable representation should include files in these formats in the representation system creating a multiple-file representation. MusicXML [28] and Capella [77] multiple-file systems for representing musical notation. However, these systems comprised files of equal format. A performance analysis representation must be able to synchronise logical descriptions of performance events with the manifestation of those events within files representing the performance.

The analysis of performance requires the logical representation of the score and performance information. Information regarding how to typeset the musical score is useful, but not essential to performance analysis, and increases complexity without significant increase in the usability in this application.

Analytical representations have valuable lessons regarding the structure of information. The kern representation represented the musical score in a truly bi-dimensional method. This is incredibly useful in the analysis of musical scores. This organisation is not possible in modern markup languages, therefore XML based representations such as MusicXML[28] and MEI[79] have adopted dual-representation systems which can prioritise the organisation of information according to time or score part. This functionality could also be provided in software within an API designed to provide easy access to the information within a representation.

The only representation which combined multiple domains of information,

SMDL [96] has been ignored by the community due to over-complexity and a lack of suitable tools for development. Therefore it is essential to create a representation in which it is easy to manipulate musical information and therefore easier to develop applications. MusicXML has very quickly become a popular representation for exchanging information between different formats. MusicXML has a very clear, user-readable format. Because it is a XML-based language it benefits from tools and APIs which have been developed for the retrieval and manipulation of information in XML documents. Many new musical formats use XML, and XSLT allows information within XML formats to be exchanged with ease. Therefore it makes sense that a format for performance analysis should be based on XML, allowing information to be imported from other formats and easing the process of developing applications. Therefore, the following chapter discusses the application of XML to the representation of musical information before a new specification for a representation is presented.

2.4 XML for the representation of music

XML is the basis for the representation presented in this thesis and for many other current representations. A thorough discussion of XML in the context of representing music is essential to enable the comparison of different representations using this standard.

XML is a meta language designed to facilitate the sharing of information between different information systems. XML is quickly becoming ubiquitous, and is a common foundation for new data representations including several which represent music [28, 79, 13, 109]. It benefits from a number of tools and libraries for verifying, accessing and manipulating information within XML documents. These tools are available on multiple platforms and programming languages. Therefore by choosing to develop a representation in XML, a host of utilities are already available. This section will discuss the merits and limitations of XML as a basis for representing musical information.

2.4.1 XSLT

The eXtensible Stylesheet Language (XSL) provides a means of describing how information in one XML language can be translated into another. The Data Format Description Language DFDL will eventually provide a means of describing the structure of binary files in XML which can be used to extract information from binary files to produce an XML document. These tools make the conversion between different formats far simpler. This makes the implementation of

translation tools for interchange formats easier, but does not help the design of an interchange format which is the limiting factor in its useful application. The ease of translation makes the use of application specific representations more feasible, rather than tailoring an application to use the structure of a particular standard, the application can use the structure most suited to the task at hand, and use XSLT to translate the content between interchange formats, or specific formats. MusicXML and MEI use this technology to provide two distinct varieties of their representation. Each version contains exactly the same content, however, one version organises information primarily by musical part, whereas the other organises information primarily by bar number. Each structure has advantages in different applications, however, using XSLT, the structure can be easily converted between the two formats.

2.4.2 Structure

XML was designed to represent structured documents and has an inherent, hierarchical organisation. The structure of a particular representation can be strictly controlled by the use of Document Type Definitions (DTD) or schemas. These provide not only a basis for implementing structure within a document, through DTDs and schemas it is possible to define the structure of documents. This allows documents to be verified to ensure that the structure and, to some degree, the content conforms to the specification. This allows a document to be verified to ensure that it conforms to the specification. These definitions support modularity within a representation, and it is possible to select/deselect parts of a specification. Elements and their contents are defined in isolation of their structural position within the overall document, therefore modularity and inheritance are supported within a document. Because document definitions may include other documents, it is possible for multiple specifications to share microstructures. This further adds to the modularity an extendibility of XML because it is possible to create a new specification which extends an existing specification without changing the original specification. The feasibility of this depends on the design of the original document, and the dependence on document validation. If the original document is not designed so as to allow modularity, inheritance and alteration may still be achieved at the expense of validation.

XML maintains a hierarchical structure: elements may not overlap i.e. element tags must be contained within the same parent element. In structural terms this means that an element can have one and only one parent. For this reason, there is some resistance towards the use of XML for representing music

as it is impossible to represent multiple overlapping hierarchies using the conventional hierarchical relationship which is dictated by the XML syntax. Figure 2.3 in Section 2.3.1 showed a very simple musical example which is reproduced in Figure 2.5 for convenience which demonstrates this problem.

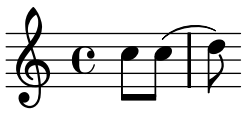


Figure 2.5: Musical example demonstrating overlapping structures

The example encoding in Figure 2.6 uses start and end tags to represent elements within a representation. The encoding could be said to be the most 'intuitive' encoding of the information in the musical example using tags, and this is the way in which text entry systems such as lilypond would represent such an example. The example is illegal within the XML syntax for two reasons:

1. The opening tag of the slur element occurs within the first bar element and closing tag occurs in the second bar element.

2. The beam and slur elements overlap.

Both reasons come from the requirement that elements must not overlap i.e. an element must have one and only one parent (excluding the root element, of course). In Figure 2.6 the tags and the indentation imply that the second note is a child element of three elements: a bar, a beam and a slur. In the lilypond example below, the containers for beaming and slurs can overlap. The left parentheses, indicating a bar occur *after* the first note which they contain. The closing parenthesis also overlaps another structure which represents a slur, indicated by the right parentheses.

c[c(] d)

This is possible because the relationship between these containers and the objects they contain is known. XML is designed to be as flexible as possible. Documents need not be described in a DTD or a Schema to be valid or useful. If overlapping elements are permitted, there is no longer any explicit definition of hierarchy within the document without a formal specification of each object and container. This also implies that it is impossible to query a particular location within such a hierarchy without parsing the entire document first.

Roland suggested that XML can describe a musical representation which is structurally isomorphic with the structural hierarchies which exist within music

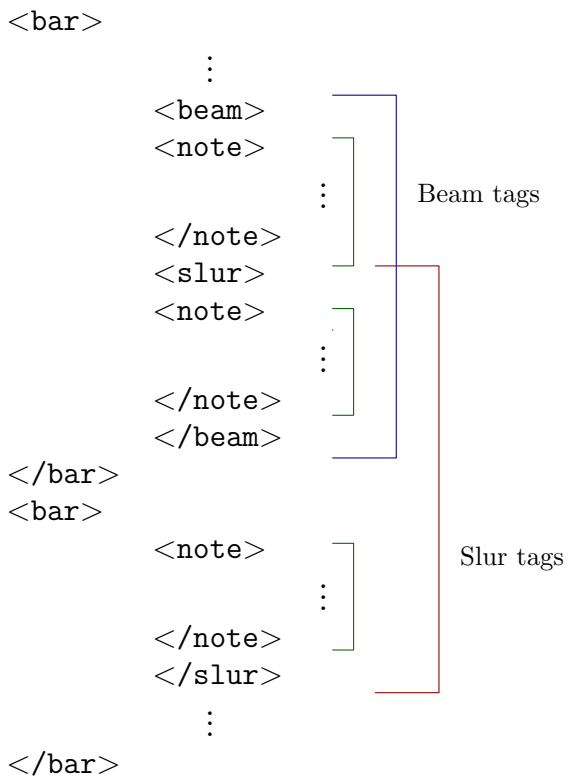


Figure 2.6: Example code describing musical example Fig.2.5 using overlapping tags.

[80]. This is certainly not the case, since multiple overlapping hierarchies exist within music, and are represented explicitly and simultaneously within western notation. The overlapping example described in Figure 2.6 is not sufficient either, because the hierarchical relationships themselves are not explicit because an element is not necessarily a child of another element because it is completely enclosed within the other element's tags.

There are some strategies commonly used to overcome XML's strict hierarchical organisation which are used in many representations, not just musical codes. One strategy uses 'milestone' elements to indicate the beginning and end of overlapping structures. This implementation, demonstrated in Figure 2.7, is used to represent overlapping structures in the same method as shown in Figure 2.6. The difference which makes this approach legal and the previous illegal, is that the start and end tags in this instance are separate elements which are related by a common id attribute. Although this is *well-formed* XML (i.e. it does not break XML syntax), it is not a good representation for a number of reasons.

Firstly, like the first example, it does not actually imply any hierarchy. There is no explicit relationship between the secondary parent (the beam) and its child objects (the notes). This means that an application cannot perform random access information retrieval on the document, because to discover whether an element is a child of a *milestone object* the parser must search through potentially the entire document to find start and/or end tags. Additionally, it is impossible to perform any validation to ensure that the milestone elements are used in a syntactically correct way (i.e. that both a start and an end tag exist, and that they appear in the correct order).

```
<bar>
        <beam type="begin"/>
        <note/>
        <note/>
        <beam type="end"/>
</bar>
```

Figure 2.7: Example code demonstrating 'milestone' elements.

Another strategy is demonstrated in (Figure 2.8) where children are grouped using common attributes. Unlike the previous example, this structure explicitly states the relationship between each note object and the beam group. Therefore it is possible to immediately determine whether a note is part of a group. However, determining the complete list of group members is less efficient than the previous example. The entire document must now be searched because there is no indication of the beginning or end of the group. The lack of any specific element means there is no logical place to define group attributes. This is contrary to the requirements of an analytical representation because it is impossible to expand the properties of groups if there is no element in which to store the information. MusicXML uses an even less useful implementation of method to indicate slurs. However, only the notes which constitute the beginning and end of a slur contain a declaration that the note is part of a slur. Therefore notes which are in the middle of a slur have no explicit relation to the group. This combines the worst features of both the solutions above.

This thesis proposes that a more suitable solution is to use relational links to define the hierarchy between objects and groups which do not fit into the primary organisational hierarchy. The example in Figure 2.9 shows the use of two way relational-links which represent the relationship between a group and its members using both the *is a member of* and *contains* relations. Thus it is possible to immediately determine whether an object belongs to another group,

```
<bar>
        <note beam="beam1" />
        <note beam="beam1"/>
</bar>
```

Figure 2.8: Example code demonstrating attribute-based bottom-up hierarchy.

and by retrieving the group element the properties and scope of the group are accessible. The existence of a group element means that the properties of the group can be extended.

The position of the group element has no effect on its properties, therefore its representation is completely explicit. This also allows the element to contribute to a completely separate hierarchy which can be defined using traditional XML parent-child/container relationship. Multiple hierarchies can be defined in this way with total independence from other hierarchies, using relational links to define relationships between notes, or other objects contained in separate hierarchies. Notes can still be contained within a traditional *staff* → *bar* or *bar* → *staff* hierarchy which allows convenient access for most purposes.

```
<bar>
     <note id="note1" beam="beam1" />
     <note id="note2" beam="beam1"/>
     <beaming>
          <beam="beam1">
               <scope notes="note1 note2"/>
          </beam>
     </beaming>
</bar>
```

Figure 2.9: Legal method of representing multiple overlapping hierarchies within XML.

MusicXML uses a compromise of the first two examples to represent some hierarchies within the standard hierarchy. Beaming is declared by the presence of milestone elements contained within the note element. Beams have a non-unique ID element which differentiates them from concurrent beams, but not from all other beams within the document. Beam elements also contain 'start', 'continue' or 'end' declaration which identifies the beginning and end of a beam. Therefore, it is immediately possible to determine that a note is part of a beam, and beam properties may be contained within one of the beam

elements. However, the complete scope of the beam element is not immediately accessible.

Relational linking is adequate when an object relates to a small number of objects. However, when an object relates to a large number of items, it may be easier to define the relation as a scope within some dimension. For example, defining the exposition can be more efficiently described by its scope within relative time rather than listing each member explicitly. A violin crescendo may be defined to have a scope within time (relative time, absolute time or both) which applies only to the violin part. It should be possible to define these references in addition to relational references for all analytical elements.

IDREF vs XPath

There are two features of XML which are suited to representing relational links. ID and IDREFs are attribute types defined in the XML standard. An ID attribute must contain an identifier which is unique in the context of all ID attributes within the document. An IDREF must contain an identifier which corresponds to an ID attribute within the document.

XPath is a language which addresses portions of an XML document. It can be used to identify elements, or ranges of elements within a document by describing position within the hierarchical structure of the document. Both alternatives have advantages and disadvantages in the context of defining relational links. ID/IDREF attributes are part of the XML specification and compliant parsers will validate ID/IDREF references. Searching by ID attribute is part of the DOM (Document Object Model) XML API specification. XPath is not part of the XML specification, and therefore cannot be validated by XML parsers.

Validation

An XML parser will identify any errors in ID/IDREF attribute definition, (whether an ID attribute is duplicated, or an IDREF does not contain a valid reference). XPath declarations are not validated by parsers. Validation is useful in self-contained documents, however, in the context of a larger system where the document is spread across several files, validation can become problematic. Where two separate performances are being analysed, the IDs of elements may need to be altered (and all corresponding IDREF attributes) to ensure individuality. Validation, however, is an optional parsing feature.

Document changes

ID/IDREF attributes are relatively robust in the face of document changes. A change in the structure, or position of elements within a document will not affect the validity of ID/IDREF links. However, if the change includes the deletion of an element which is referenced the document will become invalid.

XPath declarations are very sensitive to changes in the document structure. In some cases, the structure of the document may change without any change in content, for example a conversion from *measure* → *staff* to *staff* → *measure* organisation in MusicXML or MEI documents. In fact, the XML 1.0 specification does not guarantee that a parser will report elements in document order (it would be compliant within the specification for a parser to return the elements in alphabetical order). [65]

Finding XPath declarations which have become invalid requires an evaluation of the declaration and the changes made. In contrast, discovering ID/IDREF attributes which require alteration can be performed by validation, or simply inspecting the attribute.

A simple example of the usage of this sort of relation already in use in PML is in the description of note 'clusters'. The score-performance matcher currently used in PML projects which is discussed in chapter 3 groups performance events into clusters which are judged to have a musically simultaneous onset (chords). Each cluster contains an aggregate onset time which is derived from the onset time of all the events contained in the cluster. Therefore a cluster is a structure which contains a group of notes, and one property. The clusters are represented as a list of elements each defining one cluster. A cluster contains a list of notes which belong to the cluster identified by their ID attribute. The subelement 'onset' contains the aggregate onset for the cluster as a whole. This only use a one-way relational link from the clusters to the notes i.e. there is no explicit link within a performance note element which describes the link to a cluster element.

2.5 A Specification for the representation of musical performance

In the absence of an adequate representation for the analysis of musical performance, this section presents a specification of the requirements of a representation which aims to facilitate the analysis of musical performance issues.

```
<clusters>
         <cluster>
                 <onset ="0.23"/>
                 <scope ids="pnote1 pnote2">
         </cluster>
         <cluster>
                <onset ="0.78"/>
                 <scope ids="pnote3 pnote4">
         </cluster>
            ...
</clusters>
```

Figure 2.10: Example using relational links within PML to represent multiple hierarchies.

Information should be structured so that data is immediately accessible, and can be retrieved without the need to analyse its context. Therefore, the representation should be both declarative and explicit. The only instance where procedural information is useful is in instances where performance data is recorded as procedural information e.g. certain types of gestural information. However, the markup of this procedural information should be represented declaratively.

As explained in Section 1.2, an informed analysis of musical performance requires more than the raw performance data. In the analysis of musical performance, the score (in whatever form it may take) is necessary to evaluate the performer's expression and accuracy. Thus, a performance representation must include at least three domains of musical information: the score, the performance and the analytical domain. Each domain may contain information in a number of contexts as described in Section 1.1.

2.5.1 Score

The score may contain both graphical and logical description of score information. Of these, the logical score information is essential due to the declarative and explicit nature of the information it represents. A distinction must be made here between the logical representation of a graphical score, and the *logical score*. A logical representation of the graphical score is a declarative and explicit representation of the objects which appear on a visual score, whereas a *logical score* is a declarative and explicit representation of the *content communicated by* a graphical score. The difference can be demonstrated clearly in the representa-

tion of a tied notes. A logical representation of the graphical score will represent tied notes as two separate notes which share a common tie attribute or belong to a common group which signifies that there is a tie between the two notes. A logical score need only represent the tied notes as one distinct object. In a musical score, note start times are implicit within the context in which the note occurs. Therefore, in a logical representation of the graphical score, the temporal position of a note is not explicitly stated, but can be derived from its position relative to other objects in the score. A logical score would explicitly specify note onset time in some representation of relative, musical time.

This is not to say that the the logical representation of a visual score is of no use. The incorporation of a visual score helps visualisation at the application user interface, and is also essential for certain analyses. An adequate visual score could be derived from the logical score. However, in some cases, the logical score is not easily derived from the graphical score such as music which uses *notes inégales*.

2.5.2 Performance

Performance data covers a broad range of data types and contexts. Raw performance data exists in the phonological context as representations of the recorded sound (whether as raw audio information, or as some analytical representation such as a spectral frames), and in the gestural context as streams of motion data.

It must be possible to markup the content within these data streams to facilitate analysis. This markup constitutes a logical description of the content which can be derived from the performance data whether through automated analysis such as automatic transcription and gestural identification or through manual markup. At what point logical analyses become part of the logical domain rather than the performance domain is contentious. In this situation it becomes difficult to draw the line between what is considered analysis of performance data, and what is a logical description of raw content.

It is necessary to make such a distinction between the fundamental logical objects and their properties and information which is the results of analyses to define what information is the 'ground truth' (the term is used here 'for all practical purposes' in reality no such analyses guarantee ground truth) on which other analyses can be based. This will, in turn influence whether information is persistent or temporary within a knowledge representation. Other information

may be in the form of action lists (a MIDI file, for example), or subjective descriptions of emotive content ('vivace') or gestural content such as Laban Movement Analysis (LMA).

2.5.3 Relational links

The importance of including score information in addition to performance information has already been expressed. However, to facilitate the contextual analysis of musical performance, the mere presence of the musical score is not sufficient. The relationship between the performance and the score must be expressed in the representation. It should be possible, given a particular performance event to immediately find the musical context which may have influenced the event by analysing the relevant part of the score. Similarly, it should be possible to select a note, or a region in the score and immediately access the corresponding information in the performance. To achieve this, the score performance relationship must be described at the logical note level. A mapping between relative and absolute time, although useful for auditioning purposes and library related retrieval applications, is not an adequate representation of this relationship for analysis. A time mapping is incapable of indicating whether notated notes have been missed, un-notated notes have been added, or whether several performance notes relate to one notated note e.g. ornamentation. It also does not make the information relating to one object accessible without prior analysis to precisely locate the corresponding window in the performance. There should also be a method for creating links between performance events and the external sources such as audio files, and gestural motion streams.

2.5.4 Analysis

To facilitate analysis the representation must have a flexible, extendible and robust structure. Musical information can be described in multiple overlapping hierarchies, and the representation must support multiple, possibly overlapping organisational structures across all domains of the representation. It is convenient, for most applications, to structure musical data according to time and part. In different situations, it is convenient for either one of these to take priority. However, other situations may require organisation according to harmonic structure. Thus, multiple, simultaneous hierarchical structures must be supported.

The representation should support a modular structure which extends beyond the separation of information domains. New analyses will generate new

organisational structures, therefore it must be possible to include or reference any structural item within multiple hierarchies. The vast collection of different analytical representations necessitates modularity within the representation. Thus for any application, is should be possible to select from a set of standard modules, what information should be supplied/supported. Standardised modules also promote the shared development and use of analysis tools.

2.5.5 Deployment and Development

For any representation to survive, there must be tools which help in the creation and manipulation of data, and tools to aid the development of applications which use the representation. This fact is evident in the proliferation of representations such as Humdrum[34] & MusicXML[28] and the failure of SMDL and MEI. There is a substantial toolkit for the development of analytical systems which are based on the Humdrum representation and several repertoires of scores accessible in the Humdrum format. MusicXML quickly gained the support of the major open source and commercial notation applications and has an open API to help developers. In comparison to MusicXML, MEI, which was developed roughly parallel to MusicXML, has a more flexible structure and more thorough representation of music remains almost entirely unused because there are no available tools to import, create, or manipulate data within MEI. No software was developed even for part of the extremely complex SMDL specification. This is cited as one of the major reasons SMDL never gained sufficient user base [97].

It is important that the representation and its tools be freely distributable and that the licenses permit modification (for example GNU General Public License [27]). Thus the representation can improve and evolve to meet the requirements of the users. This includes the applications and APIs which are based on the representation. Without open source tools, developing a system or application to perform analysis may require the development of enabling technologies such as score-performance matchers, notation rendering and audio analysis tools.

2.5.6 Summary

1. Information should be explicit and declarative.
2. The following three contexts should each be adequately represented:
   (a) Score
   (b) Performance (including gesture)

(c) Analysis

3. The correlation between score and performance representations must be represented at the event level.

4. The representation must support multiple hierarchical structures simultaneously.

5. It must support cross-domain structural relations.

6. The representation must be extendable and modular.

7. The representation must be freely distributable and modifiable.

8. There must be available tools to assist in the gathering, manipulation and analysis of data.

Existing systems can be analysed in the context of this specification. SMDL passes the 2nd and 3rd criteria. It supports multiple domains (though in practice, only the logical score domain was specified) and cross-domain relations but SMDL lacks the support of any software and is based on SGML, a technology which has been more or less superceded by XML. Furthermore, SMDL is now considered a dead language [36].

MusicXML satisfies many of the criteria. It represents information explicitly and declaratively with the exception of relative onset times of notes. It is freely distributable, and being based on XML can be extended in a modular fashion. Many tools exist for the creation and manipulation of information within MusicXML, including several tools to exchange information between various other formats. However, it has no support for domains of information other than the musical score. The use of ID attributes (discussed in Section 2.4) within note elements would permit the representation of relational links including score-performance correlations if the representation was extended or included within a mixed-representational system. The representation described by Hirata et al. [35] which extends MusicXML only permits the inclusion of MIDI data within the structure of MusicXML. It therefore fails to support multiple simultaneous hierarchies, and MIDI data is an inadequate description of performance information for analysis.

MEI also conforms to many of the criteria. It represents multiple domains, but only within the hierarchy designed to represent the musical score, and not in separate, multiple hierarchies. MEI, like MusicXML, could be extended to include adequate support for other hierarchies, however it fails on the last point

in that despite its maturity as a language, there are no applications which support it.

There are no representations which completely conform to this specification. For this reason, Performance Markup Language (PML) was developed. The following section presents PML and will discuss the design methodology and structure of PML.

2.6 PML

PML has been designed by the author of this thesis to form the basis for a complete representation of musical performance for the analysis of performance issues. It is a specification for such a representation, rather than a complete representation in itself, which relies on the integration with a separate notation representation for score description. This section will describe the structure and features of the specification. A Appendix A contains and annotated version of the DTD which defines the specification.

2.6.1 PML Document

The basic structure of a PML document consists of three parts: score, performance and analysis. The document must contain one score. The definition of the score hierarchy is not part of the PML specification, and it is theoretically possible for PML to be used in conjunction with any XML-based score representation. The reasons for this are twofold. The complete absence of a useful open API for the display of musical notation means that the incorporation of an existing representation which has the support of existing applications makes PML instantly accessible. Secondly, the ability to use any XML-based score representation means that it can be used to extend existing corpora of musical scores. Figure 2.11 demonstrates the basic hierarchy of PML as a block diagram and figure 2.12 describes the same structure in basic XML markup.

A PML document may contain one or more *performance* elements thus facilitating the comparison of different performances of the same score. A performance element contains markup of the fundamental logical performance events. The final section of a PML document contains analytical structures.

Using XML entity references, document fragments may be included in separate files allowing the same score file or performance markup to be used in multiple PML documents. Therefore there is no need for the duplication of data when analysing a performance in isolation and in the context of other performances and increases maintainability an consistency of results.

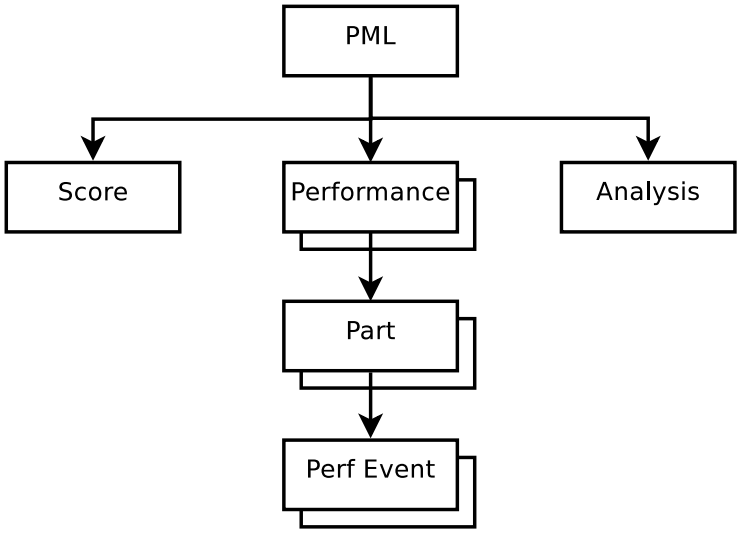


Figure 2.11: Block diagram describing structure of PML specification

```
<pml>
      <score>
      ⋮
      </score>
      <performance/>
            <perfpart>
                  <event>
                  </event>
                  ⋮
            </perfpart>
            ⋮
      </performance>
      <analysis>
      ⋮
      </analysis>
</pml>
```

Figure 2.12: Basic structure of a PML document, expressed using XML tags

2.6.2 Performance

The *Performance* element contains all the information which is pertinent to a single performance. It must include one or more *Performance Parts* and may include analytical hierarchies which relate solely to the performance and data contained herein. In addition, the performance element should contain metadata regarding the overall performance such as the time, date, location of the performance, the recording engineer. This is also where a reference frequency and corresponding pitch should be recorded should the performance be performed in anything other than concert pitch (A above middle C = 440Hz). This is required to ensure accurate conversion between frequency and pitch.

2.6.3 Performance Part

A performance part contains information regarding the performance of one performer. It should contain a markup of the logical properties of performance events. These events are subdivided into *sound events* and *gestural events*. The elements which contain the events should also contain metadata concerning the method which was used to determine the events and their properties. Therefore it is possible that there may be more than one *gestural events* element as different gestures may be detected using a variety of techniques and sources. It is therefore also possible, though less likely that there may need to be multiple *sound events* elements.

The performance part should also contain metadata relating to the individual performer and their performance. Therefore, performer metadata and important information regarding the instrument and the capture of information relating solely to this instrument such as the microphone used for recording should be noted here. In addition, if the events within a container all use a common time base (see below), this can be declared within the container element. However, if events within a single container are extracted using a common algorithm, it is likely that a common timebase will be used.

2.6.4 Event

Sound events and *gestural events* describe events within a performance part. Every event must contain a start time and, optionally an end time. Time can be specified in seconds, or in the time base of the source medium from which the properties of the event were derived. Therefore the onset of a gestural event can be defined as occurring simultaneously with an exact frame of a video stream, whereas an audio event can be related to a sample frame within the time base

of an audio file which has a completely different frame rate. Relating an event to the timebase of an external entity is achieved using an external reference element (see section 2.6.6).

Events should contain other observable properties which are unique to the event such as dynamics or fundamental frequency. A sound event may contain both the performed frequency and the performed pitch or neither. Pitch can be denoted as either a pitch number (which specifies the sounding note, but not the enharmonic equivalent) or a complete pitch description. Where pitch information is ambiguous, the ambiguity must be preserved e.g. when performed pitch is derived from MIDI data.

Events are aligned to notes within the score using the *align* tag. The tag will specify a link to one note by reference to a note's ID, or in later versions, an XPath expression. This tag can also specify a *repeat* number which distinguishes between instances of the same notated note at different iterations of repeated sections of the piece.

Figure 2.13 contains some example XML markup in PML which expands upon the example in figure 2.12. The *score* element now contains a *scorepart* element which in turn contains a *note* element. The scorepart element is identified by the *id* attribute (of type `ID`) which has a value of "p1". The *perfpart* element in the performance references the relevant part in the score using the *part* attribute (of type `IDREF`). The event in the performance corresponds to the note in the score. This is indicated by the *align* element which references the note in the score by IDREF. The onset and end times for the event are specified in seconds, and the frequency in hertz.

2.6.5 Gesture

The gestural events currently have no content specification. Research into physical gestures in musical performance covers a wide range of divergent topics, and this is reflected on the wide variety of gestural data, capture methods and the fundamental logical events which are the focus of the research. Examples include identifying circular motions in the movement of the bell of a clarinet[108], or identifying fingering in piano performances. A unified method of describing these gestures has yet to be agreed, though with a common representation such as PML, captured data, tools and analytical methods and results can be shared easily between research groups, encouraging the formation of a unified ontology for the field including the representation of data streams and common gestural 'scenes'. A discussion has already taken place to begin the process of unifying the community and creating a common representation and common tools [46].

```
<score>
      <scorepart id="p1">
            <note id="note1">
            ⋮
            </note>
            ⋮
      </scorepart>
</score>
<performance>
      <perfpart part="p1">
            <event id="pnote1">
                  <onset>0.145</onset>
                  <end>0.564</end>
                  <freq>440</freq>
                  <align note="note1"/>
            </event>
            ⋮
      </perfpart>
</performance>
```

Figure 2.13: PML performance structure in XML

2.6.6 Cross referencing

Cross referencing within and outwith the XML document is currently facilitated using three methods:

Align The *align* element defines the link between a performance element and its corresponding score element.

Extref The *extref* & *extres* element 'groups' describe a reference to a position within an external resource (i.e. any non-XML file format).

Scope The *scope* element allows a group of objects to be defined.

The align element, described above defines the links between the score and the performance. This is currently a mono-directional relation. There are two reasons for choosing this in the current implementation. Firstly, to create bi-directional links would require a change in the MusicXML DTD. The MusicXML DTD is designed in such a way that prohibits making changes without creating a separate DTD. If the original DTD is included as an entity, a redefinition of the element content descriptions causes a parsing error. Secondly, by leaving the structure of the original score intact, all PML files analysing the same piece can use the same score file as an external entity, ensuring consistency and maximising storage efficiency.

References to external sources are described using two elements. The external resource elements describe a file itself, and the external reference elements point to a particular position within an external resource. These two element types must be described for each external file format which is supported by the DTD. An external resource element contains a reference to an external entity which declares the actual file and its location. It may also contain default attributes which indicate default attributes for accessing locations within a file, for example it may specify a particular channel in a multitrack audio file which will be used in the absence of an overriding attribute in an external reference. It should also contain sufficient information to convert the information contained within the external reference element into other related time formats without requiring an examination of the file itself. Therefore, the definition of a WAV file contains the frame rate, which will allow any reference to a particular frame within the file to be converted into seconds, or related to the closest frame in another resource with a different frame rate.

External references must reference an external resource element (using an ID/IDREF relation), and contain information to reference a particular point within that resource. Where the external resource element has specified a default

property, this will be overridden by any similar property defined in this element. Therefore, many external references will relate to a single external resource. Also, it is possible to create more than one resource which relates to a single file, for example a large GMS file may be divided into separate gestural scenes, or a multitrack audio file might be declared as separate resources which relate to recordings of separate instruments.

The *scope* element allows groups of elements and ranges of data to be defined. Groups can be defined by either listing all the elements of a group (2.14) or defining a range in which those elements occur. Lists of objects are referenced by their `ID` attributes in the refs attribute (which is of type IDREFS). An example of this was shown in Figure 2.10. A range is specified by the elements `from` and `to`. These elements may contain any number of external references or references by `ID`. Therefore, a phrase within a piece of music may be described as a group of notes in the score, identified by their `ID`, as a group of events within performance markup, as a range in time in seconds (figure 2.16), or as a range within an external source such as an audio file as shown in 2.17. Alternatively, a range may also be specified by using the `ID` attributes of the first and last objects as in figure 2.15.

```
<scope refs="note1 note2 note3 note4"/>
```

Figure 2.14: Scope referencing notes by IDREF list.

```
<scope>
      <from refs="note1"/>
      <to refs="note4"/>
</scope>
```

Figure 2.15: Scope referencing a range of notes using IDREFs.

2.6.7 Analytical Structures

Specific analytical elements have not been defined, although the scope element can be used to describe complex relations between the score, performance events, external sources or other structures within the analytical domain. Naturally, hierarchy within an an analytical organisation can be expressed using the traditional XML parent & child object association. However, the scope element described above allows relations which can't be described using standard XML

```
<scope>
       <from>
              <atime>0.564</atime>
       </from>
       <to>
              <atime>0.9</atime>
       </to>
</scope>
```

Figure 2.16: Scope referencing a region in performance using seconds.

```
<wavres
       id="wav01"
       file="performance.wav"
       fps="44100"
       channels="1" />
:<scope>
       <from>
              <wavref wavres="wav01"
                      channel="1"
                      frame="24872" />
       </from>
       <to>
              <wavref wavres="wav01"
                      channel="1"
                      frame="39690" />
       </to>
</scope>
```

Figure 2.17: Scope referencing a region in an external resource.

hierarchy such as overlapping hierarchies (by allowing several hierarchies to reference the same data).

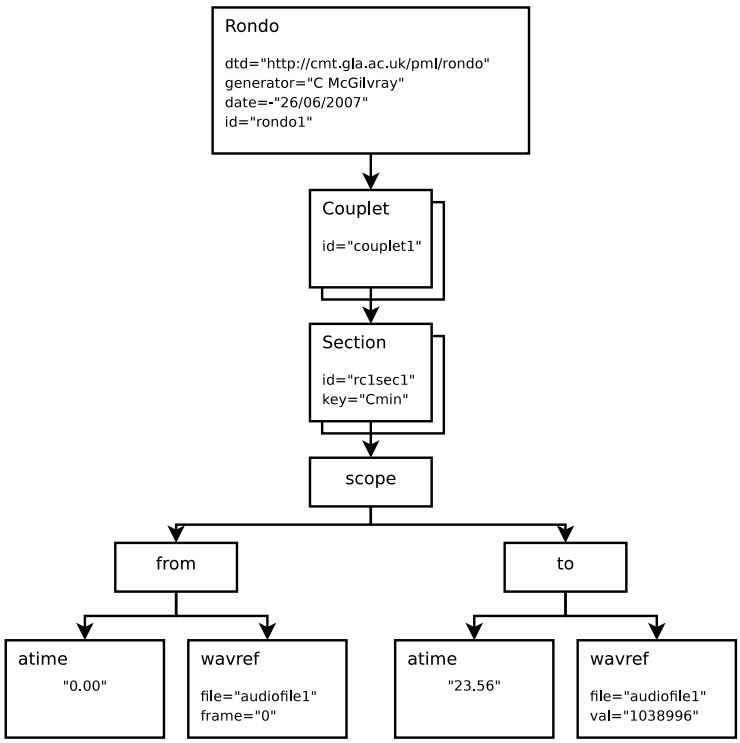


Figure 2.18: Possible analytical hierarchy describing structure of a rondo.

These only provide a method to describe the relation between elements. The description of the semantics of the relation should be defined in the specification of the additional analytical structures. These specifications should describe the entire analytical structure as it relates to the PML structure. For example a markup of the high level structure of a rondo could be strictly defined as a sequence of couplets, each couplet contains a description of two sections by their range in relative and absolute time and the key within that section. This hierarchy should be contained within a single element which also contains metadata to describe the process by which the data and structures were derived. For example, should the analysis have been performed by hand, the metadata should include the name of the editor. If the analyses were automatically generated, the algorithm and the exact version number of the algorithm should be identified. Formally describing the hierarchy encourages the formation of standard descriptions of musical structure and allows common tools to be developed. A description of the algorithm ensures that the validity of the data can be checked

and allows different analysis methods to be compared. An example hierarchy can be seen in Figure 2.18.

This hierarchy is a possible analytical structure representing the structure of a rondo in the context of both the score and the performance. The scope of each section is described in terms of relative time and absolute time. The absolute time element *atime* contains both a value, and a reference to an element which describes an external source. In this case, the source is an audio file, and the element will describe the properties of the audio file, such as frames per second and number of channels which not only allows the range to be selected from the file, but allows the time to be converted to any other timebase. The exact score and performance to which the analysis refers is indicated by IDREF attributes. This is vital when the document contains information on multiple performances. The basic structure here might be extended to include further descriptions such as codas, codettas, or a distinction between variations of rondo form.

2.6.8 Tuning

PML contains support for the definition of a tuning system within a PML document. The tuning description resides within the PML 'performance' element as it provides the means to translate between notated pitch, frequency and MIDI number. The tuning definition is taken from the development of the microtonal extensions to the Rosegarden sequencer, part of the Microtonalism project discussed in Chapter 4.

The very compact description contains a list of intervals, which may be denoted either in cents or integer ratios, an ordered set of lists of enharmonically equivalent spellings (i.e. for each interval there exists a list of all corresponding pitch spellings), and a set of references which allow pitch, frequency and MIDI to be mapped.

- Root pitch: The root pitch determines the root of a non-equal tempered scale i.e. the pitch which represents interval 0. This is only required for non-equally tempered tunings systems.

- Reference Pitch: The reference pitch maps between pitch and frequency. It contains one pitch and one frequency in Hertz. From this one reference, the entire map of intervals and frequencies can be generated. The default if no reference is specified is A4 = 440Hz.

- Midi Reference: The MIDI reference maps pitches to integer notation. This contains one pitch and one integer value. This can be used to map

between MIDI values or any other integer pitch notation and frequency or pitch. The default behaviour in the absence of any definition is to map C3 to the number 60.

Figures 2.19 & 2.20 together demonstrate a description of a 19-tone equal tempered tuning. Figure 2.20 defines the references explained above and the list of intervals in the tuning system. The reference pitch A4 is set to 440Hz (this is the default, and therefore need not be included). The midi reference note is note number 69 and is set to represent A4. The root spelling is defined as C, however because this is an equal-tempered tuning, this has no effect. In any case, C is the default and therefore this definition may also be excluded. The interval list defines from a reference pitch (and in this case the interval between each is one 19^{th} of an octave).

The tuning definition continues in figure 2.20. The spelling list defines equivalent spellings for the tuning. In this tuning, C# and D♭ represent separate intervals in the scale, while E# and F♭ represent the same interval between E and F.

Obviously, in the conversion to pitch from either MIDI or frequency, the exact enharmonic spelling cannot be obtained unless only one pitch spelling corresponds to that interval within the system or an algorithm is used to derive the spelling from intonation or harmonic context. In the absence of any definition of a tuning system, the default is to assume a 12 tone equal tempered tuning system where A4 corresponds to 440Hz and C3 relates to integer pitch 60.

2.6.9 Separation of logical and analytical information

It is encouraged that analytical information should be separated from logical information wherever possible. Although PML can theoretically be expanded at any level within the document, it is recommended that analytical structures be contained within the analysis element which is the direct child of the root PML element.

However, the logical descriptions of performance events can rarely be considered 'ground truth' and are in fact the results of analyses themselves. Nonetheless, in the interests of practicality, a distinction must be made otherwise research could never proceed. The fundamental logical objects will be the basis of all analyses, whereas secondary analyses will be required less frequently and are also more susceptible to alternative interpretations. An object can be said to be a fundamental logical event if it conforms to the following two conditions:

```
<tuning name="19ET">
       <refpitch>
              <pitch note="A" octave"4"/>
              <freq>440</freq>
       </refpitch>
       <midiref>
              <pitch note="A" octave="4"/>
              <midi>69</midi>
       </midiref>
       <rootpitch>
              <spelling note="C"/>
       </rootpitch>
       <intervallist>
              <interval>63.158</interval>
              <interval>126.31</interval>
              <interval>189.47</interval>
              <interval>252.63</interval>
              ⋮
              <interval>1200.00</interval>
       </intervallist>
       ⋮
```

Figure 2.19: Example tuning definition in PML describing 19-tone equal tempered tuning (continued in figure 2.20)

```
⋮
<spellinglist>
       <enharmequiv>
              <spelling note="C"/>
       </enharmequiv>
       <enharmequiv>
              <spelling acc="Sharp" note="C"/>
       </enharmequiv>
       <enharmequiv>
              <spelling acc="Flat" note="D"/>
       </enharmequiv>
       <enharmequiv>
              <spelling note="D"/>
       </enharmequiv>
       ⋮
       <enharmequiv>
              <spelling acc="Sharp" note="E"/>
              <spelling acc="Flat" note="F"/>
       </enharmequiv>
       ⋮
</spellinglist> </tuning>
```

Figure 2.20: Example tuning definition in PML describing 19-tone equal tempered tuning continued from figure 2.19

Observability The object described is observable within the raw performance data and can thus be described as something which exists within the time space of one of the performance data streams.

Atomicity The object can be described as a single entity which is not the sum of smaller subelements.

These fundamental descriptions are contained within the performance element. The descriptions are further separated into those which describe sound events, and those which describe gestural events. Sound events naturally describe audible events such as the sound of a guitar being plucked and contain properties such as onset, duration & fundamental frequency. Gestural events describe physical actions which accompany the performance. A gestural description of the previous example may include multiple events such as the depression of the string which includes the onset, duration, fret position and finger used to make the gesture. This would be accompanied by an element which describes the plucking gesture with properties such as onset (but no duration), string and finger. It should be noted, that although the sound event is the result of multiple elements within the gestural domain, the audible artefact remains an isolated entity. Even when the sound event is derived from gestural information rather than audio data (e.g. the use of MIDI data to derive logical properties of sound events) the event should be described as a fundamental logical object (the means by which the sound events were derived should, in any case be described in the metadata which accompanies the events).

Structure Repetition

Containing analysis structures within the structure of a performance can express the scope of the analysis for example location analysis information within a performance part limits the scope of the analysis to a particular part within a particular performance. It also makes it immediately apparent what data should be included when compiling a document based on multiple performances and their associated analyses. If analytical objects arranged according to performance & performance part are represented within the analytical domain it could lead to the unnecessary replication of a structure already defined as part of the performance domain.

However, locating analyses within performance structures can also cause repetition, especially when multiple performances are contained within a document. There is no 'one size fits all' solution to this problem as long as a

strict hierarchical system is being used to represent a multiply hierarchical data model.

Locating all analysis structures within the global analysis element reinforces the separation between the score, performance and analytical domains. It eases the selection of analytical data based on the properties of the analytical process and the integration of various modules of analytical data. For these reasons it is encouraged, though not enforced that analytical information be located within a separate domain.

Extendibility & Compatibility

Several strategies have been employed in the design of the PML DTD to ensure that the representation is easily extendible while maintaining compatibility and validity. Validity is not a requirement for an XML representation. Validation ensures that the structure and content conforms to some defined standard. However, a document may be 'well formed' i.e. the document is legal and the syntax fully conforms to the XML standard. There are two main advantages to validation in the context of PML. Firstly, validation encourages the development and use of standard representational structures. It is a long term goal of PML that modular extensions be developed to represent different aspects of musical performance. Secondly, validation identifies where content does not conform to the specification. This includes checking the validity of ID/IDREF attributes. These references are integral to PML's multiply hierarchical structure. If PML files are to be constructed using several separate files, it may become difficult to maintain these relationships without the ability to highlight invalid relations which can be performed using any validating XML parser.

The amalgamation of different representations can cause several problems if validation is required. The structure of a document is defined by specifying the contents of each element based on its tag name. Therefore each tag name can be associated with only one content description. In the case where two DTD's are combined, this can cause naming conflicts. For example, PML contains an 'offset' element which describes the time at which a performance event ends. However, MusicXML uses an element named 'offset' to describe the graphical placement of certain symbols relative to a particular time. In this instance, the specification of the *type* of data is the same: both elements contain only parsed character data (PCDATA) although the actual *meaning* is completely different). A validating parser will report an error because the offset element has been declared twice. In the event that the processing continued, each offset element would be validated against the declaration which occurred latest in the

DTD, therefore if the specification differed, the document would be invalid.

Although the development of XML Namespaces began before the completion of XML1.0 [106], it is widely considered that namespaces have been implemented very poorly, and their inclusion is certainly controversial [99]. Namespaces were introduced to resolve naming conflicts between elements of different representations. Naming conflicts caused problems not only in validation, but had the possibility to cause confusion in applications which may not know how to process information contained within an element if it is not clear to which representational scheme the element belongs.

However, many representations (including at the time of writing MusicXML [28], Wedelmusic[109], MEI[79]) have no support for namespaces. This may be simply because namespaces are considered to be poorly implemented. One example of this is the difficulty with which namespaces can be used within a system which validates its documents using DTDs. In fact, because of this, it is a common view that the benefits of namespaces and document validation are often mutually exclusive. The two examples in 2.21(a) show how namespaces are used to distinguish between representations using a simplified view of the first structural level of PML. The first example contains no use of namespaces. The second example 2.21(b) shows the use of the namespace prefix 'pml' to distinguish between elements which are a part of PML, and those which belong to the externally defined score cluster.

```
<pml>
      <score/>
      <performance/>
</pml>
```

(a) Without a namespace prefix.

```
<pml:pml>
      <score/>
      <pml:performance/>
</pml:pml>
```

(b) With namespace prefix 'pml'. (The namespace declaration is omitted for conciseness)

Figure 2.21: Demonstrating XML namespace prefixes.

Within a DTD, there is no link between the namespace prefix and the Uniform Resource Identifier (URI). Therefore validating parsers treat a namespace prefix as a part of the element tag name. Consequently, if the DTD does not include the exact prefix used in the XML document within the tag name in the DTD, the document will not be valid[1]. The majority, if not all XML score representations make no use of namespaces within their DTDs. Therefore namespaces cannot be used to identify elements within the score structure. The only

[1]This is only true for DTD-based validation, the various schema based language descriptions have a more sophisticated handling of namespaces

solution is to adopt the scheme used in 2.21(b) in which the score representation belongs to no namespace. In this instance the namespace of PML (and any other representation apart from the score representation) must be specified explicitly for *every* element which belongs to the PML DTD.

Namespaces can also cause conflicts where two representations use the same prefix. Supposing another representation called Percussion Markup Language was to be integrated with PML. For example, 'Procedural Markup Language' [75] describes multimedia content and presentation information and shares the same abbreviated name as PML. It is possible that both representations may have defined the same namespace prefix. Therefore, if the two representations are to be integrated into one valid document, it must be possible to change the prefix used for PML elements.

This can be achieved by defining the namespace prefix in the DTD within a *parameter entity*. Parameter entities are used to define reusable content within DTDs. They associate text which will be substituted wherever the entity is used within the DTD. The text can contain lists or fragments of declarations. They can also contain other parameter entities which will be expanded by the parser. This provides the ability to create reusable declarations, or construct a hierarchy of inheritance within the content specification. Parameter entities can also be redefined allowing modification to the contents of an external DTD which has been included in another. Below is the declaration of the default namespace prefix within PML.

```
<!ENTITY % nsp 'pml' >
```

This can be combined with an element tag within another parameter entity as below to create the tag <pml:analysis>. Combining the prefix and the tag in this way further litters the DTD, but this is necessary to avoid the addition white space around the substituted text of the prefix entity. The resulting parameter entity is then used to reference the element within the specification. The next line shows the declaration of the pml element and its contents.

```
<!ENTITY  %analysis %nsp;:analysis>
<!ELEMENT %pml;
          ( %score; , %performance;* , %analysis;?  )  >
```

The default prefix can then be changed by including the original PML DTD within another DTD after redeclaring the namespace prefix entity. The simple example below renames the prefix to 'perfml'.

```
<!ENTITY % nsp 'perfml' >
<!ENTITY % pmldtd
          SYSTEM 'http://n-ism.org/pml/pml.dtd' > %pmldtd;
```

Parameter entities are also used to allow the content of PML elements to be extended while maintaining a fully valid DTD. The specification for DTD's permits extension of element content through the use of the *ANY* content model. This allows the structure of the element to contain any combination of elements *which have already been declared*, in any order at this point. Extra elements can be declared in an external document which references the DTD in a similar method used to redeclare the namespace prefix above. Although this provides a simple and flexible method of extending the DTD, it allows no formal description of the structure of the extensions. An alternative is to include an empty parameter entity within the content specification of each element. The empty parameter entity makes no change to the content specification of the element. Therefore elements cannot be added without modification to the DTD, and the structure of the extension can be formally specified. This constraint again reinforces the use of standard extensions, although the parameter entity could still be redeclared to contain the *ANY* content model until such time as the extensions have been standardised.

MEI uses parameter entities to define the tag names of elements within the representation. The intended purpose of this is not clear from the documentation included within the DTD. Although, it does allow the tag name of each element to be changed individually. Changing the name of an individual element would solve a naming conflict if MEI were to be used with another representation, although changing individual elements causes inconsistent changes which would eventually result in many variations of MEI with modifications which, although minor, would cause invalidity and incompatibility with processors. It is possible to add a namespace prefix across the entire structure however this would require a redeclaration of every tag name within the representation. MEIs use of parameter entities to describe reusable content descriptions permits extension through redeclaration, however, entities are only used for common content descriptions therefore individual elements cannot be modified. MusicXML makes very limited use of parameter entities and only as reusable content models.

These restrictions place a slight burden on developers. The cost of ensuring namespaces are explicitly specified is insignificant compared to the power of

identifying invalid ID/IDREFS through validation. The restrictions placed on extension enforce modification to the DTD to allow a valid document, but this encourages an examination of the document structure before modifications are made. It also promotes the use of formalised structures which can be compared, shared and used to promote the development of community wide standards.

2.6.10 PML: Current implementation and Software

The current implementation of PML which is in used in the Centre for Music Technology, University of Glasgow uses MusicXML to represent musical score information. The DTD of the current PML implementation includes the MusicXML DTD as an external entity. This allows the implementation to be completely formalised, and permits validation of documents.

There are many aspects of MusicXML which, although adequate for the interchange of musical scores, are impractical for the representation of a musical score for analysis. This theses does not claim that MusicXML is an optimal solution even for the application for which it was designed (the interchange of graphical musical scores), and certainly does not claim that it is adequate for the purposes of analysis. It is however, currently the optimal solution in a practical sense. XML is a human readable format, but this is for the benefit of the developer, not to the user. Verbosity helps identify the structure of XML documents but does not necessarily make it easier to edit the data by hand, especially in data-centric XML documents. Therefore, applications must be available to facilitate the creation and edition of score files. MusicXML has considerably larger list of compatible applications than any other XML-based score format (including some open-source applications [63, 81]). There is also an open-source library for the manipulation of MusicXML documents, and an on-line repository for MusicXML scores. The software library can be used in the development of applications which support MusicXML-based PML.

Several tools have been developed to support the use of PML within projects based at the Centre for Music Technology (CMT), University of Glasgow and *n*-ISM [61]. These include the development of an API for the creation and manipulation of PML documents which has been developed along with all other tools.

The most significant tool is the polyphonic score-performance matching algorithm described in Section 3. This matcher achieves very accurate and robust matching by using score information to dynamically adapt the matching process. This matcher can only operate with a representation which is capable of representing the score and performance simultaneously. Similarly, such a rep-

resentation is only of use if it contains the event level correspondence between score and performance events which this type of matcher provides. Other tools include:

mxml2pml is a tool for the validation & preparation of MusicXML files. e.g. ensuring that notes within MusicXML all contain ID attributes.

MIDI2pml creates performance events based on the information MIDI files.

aud2pml, this tool allows the manual markup of performance events. Events can be labelled using the Audacity audio editor [3] and these labels used to create events within a performance part in a PML file.

pmlPitchAnal was developed to examine pitch trajectories of performance events. It uses the pitch tracking algorithms of the Praat speech analysis program [7] to extract and visualise the trajectories of performance events. It was developed for the analysis of pitch trajectories in sprechstimme.

Additionally, a system has been developed for presenting analytical information as an annotation to a musical score. Information such as deviation from the notated pitch can be displayed alongside the score, providing user-friendly feedback regarding aspects of a performance. In the example in Figure 2.22, the bars represent a comparison of note duration between two different performances of the same piece.

The remainder of this document will present research which has been conducted using PML. The following chapter describes the polyphonic matching algorithm which has been developed to create the relationships between score and performance necessary of such a representation. Finally a description of two ongoing projects which use PML for the analysis of pitch accuracy and intonation. The first project, *Microtonalism* [31], addresses issues in the rehearsal of microtonal music. In this project, PML has been used to assess the pitch accuracy of a microtonal performance. In the final project which will be presented here, PML has been used in the investigation of intonation in the performance of Pierrot Lunaire.

Other projects include the multi-modal performance analysis of the finale in Chopin's B♭ minor Sonata opus 35,. which includes the analysis of audio, video tracking and physical gestures[45]. Also, PML is being used as the external format for at least one large-scale database of musical performances.

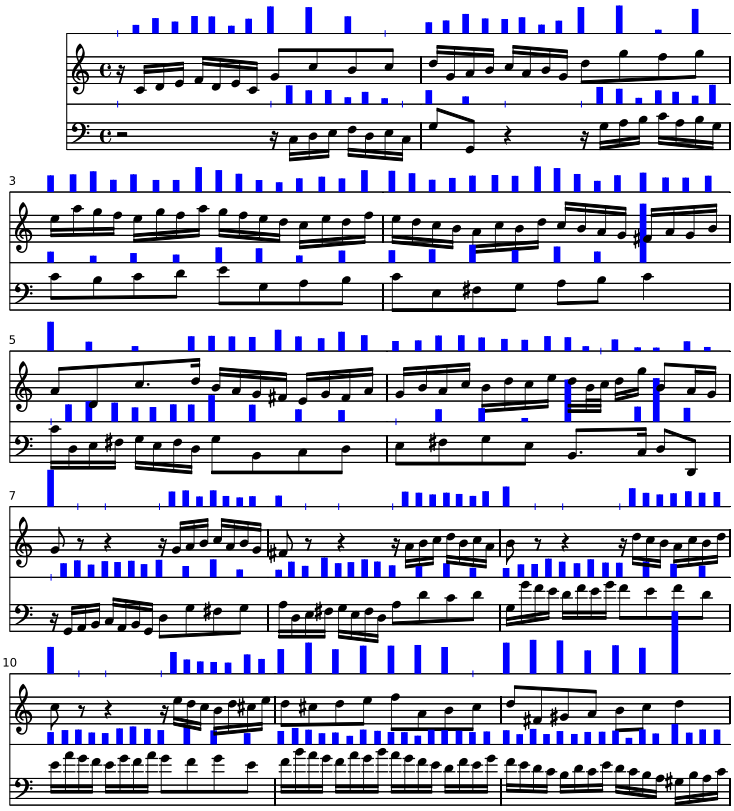


Figure 2.22: A musical score annotated with analytical information derived using PML. The bars display gestural information: the duration of key presses in a performance of Bach's Invention No. 1.

2.7 Conclusion

There are many issues associated with the representation of musical performance. In light of these issues, the suitability of various methodologies & technologies have been discussed and their suitability towards the representation of musical performance have been assessed in chapter 2. This has lead to the formation of a specification for such a representation. This chapter has presented an implementation of this specification

The current implementation currently fulfils all the criteria set forth in section 2.5.6. However, there are areas mentioned in the criteria which have considerable room for improvement.

The representation would benefit from a more suitable score representation. For example, a truly vertical structure which represents information in concurrent slices of time would be beneficial for harmonic analysis. However, the PML

specification can be used to extend any XML score representation. MusicXML is merely the most practical representation to use at the current time. For example, the onset time of notational objects in MusicXML is implicit in the order of elements within the score. Including an element to explicitly declare the relative time means the score will no longer validate against the standard MusicXML DTD. This can be overcome in the API, however this is not ideal. When a suitable alternative representation and the tools to manipulate it become available, PML can be used as an extension for performance analysis using that alternative. The representation of performance and analytical information should not change, and only the code within analytical algorithms which accesses the score domain needs to be altered to use another representation. In this light, application authors should attempt to refrain from using structures or information which is specific to MusicXML, (or any other score representation) hence the independent definition of relative time within PML. Thus transferring applications to other incarnations of PML which utilise a different score representation will only require changing the code which accesses score information. Consequently, the analytical representations and algorithms will remain valid.

The specification of analytical structures and gestural information is currently lacking within the representation. However, the academic community has still to decide exactly how this information should be represented [46]. Only once performance issues have been thoroughly explored can suitable generic representations be created for these areas.

Finally, the representation will benefit from the further development of applications and utilities to populate and manipulate the representation. As stated earlier, a lack of adequate tools will prevent a representation from gaining support. The use of MusicXML as the score representation means that a host of existing applications can be used in the manipulation of scores such as:

- *NoteEdit* [63], an open source notation editor.
- *xml2ly* [119], a MusicXML to Lilypond conversion tool which can produce beautifully typeset scores.
- *Rosegarden* [81], an open source MIDI sequencer with notational capabilities. This has also been modified as part of a project discussed later in Chapter 4 to support microtonal capabilities.

Chapter 3

Score-Performance Matching

The first section will explain the difference between different types of matchers in the musical domain and musical applications in which they are employed. Following on from this, the technologies which are used to implement the majority of matching algorithms will be examined discussing existing algorithms which utilise these technologies.

The second section will examine one of these technologies, Dynamic Programming (DP). There are many subtle differences between the ways in which this technology has been implemented in musical matching algorithms. Each potential variable will be discussed including the effect these have on the operation & output of the matcher in the context of musical applications.

The third section discusses the use of DP in the context of polyphonic matching and describes a novel method of applying DP to this task. A brief section discussing how performance of matchers can be evaluated follows.

The fifth section provides novel solutions to some of the deficiencies of the DP method. The affect these changes have on the accuracy of the matcher first presented in the previous chapter are examined incrementally.

Finally, the success of the improvements will be discussed, and suggestions made for continuing the development of such an algorithm.

3.1 What is score matching?

Score performance matching is part of a family of closely related tasks which aim to find temporal correlations between two different musical objects. Each

task within the field can be described by the type of data on which it operates and the type of correlations which are identified between the two objects.

The objects of analysis may consist of different domains of musical information. The data may be symbolic musical information i.e. a computer representation of a musical score which represents time and pitch as relative values. The information relate to the performance such as logical performance information described in terms of onset, duration and frequency or pitch, for example, information derived from MIDI information [21]. Some systems match a performance in using features extracted from audio such as spectral analyses, fundamental frequency [42, 66, 12] or even raw audio data itself [76].

The correlation between objects can consist of event-level correspondence, or temporal correlation. Event-level correspondence requires that each object to consist of a list of events. Each event within an object can be shown to correspond to an event or events in the other object. A corresponding event need not be representative of the event to which it corresponds. A correspondence between two unrepresentative objects may indicate a deviation from the reference object e.g. a performance error, otherwise known as a substitution or, a 'wrong' note.

Temporal correlation derives a map between the time-lines of two objects. This may be between objects with absolute or symbolic information. It is possible to derive a temporal correlation from event-level correspondence. However, whereas objects may occur simultaneously in relative time, in absolute time, performed events do not occur at exactly the same time. Resolving the onsets of a group of notes into a single point in time may be achieved in a number of ways by using for example:

- The first or last onset in the cluster
- The median of all onsets
- The mean of each of the onset
- The onsets of surrounding notes or note clusters

The chosen method may depend on the style of the performer, the style of the piece, or the musical context in which the notes appear.

The term 'alignment' has been used in different circumstances to refer to either temporal correlation or event level correspondence. For the purposes of this thesis, event-level correspondence will be referred to as matching, and temporal correspondence will be termed alignment.

3.1.1 Types of matchers

Score Matching

Score matching aims to analyse two symbolic musical scores identifying correspondences at the event level. The most active application of this technology is in the field of music information retrieval. Matching entire musical scores can help identify differences between various editions of the same piece. The process of matching score excerpts can be used for many tasks relating to music retrieval. A complete piece of music can be retrieved from large databases based on a small search query [62]. Thematic, harmonic or rhythmic content can be found within individual pieces. To find and rate inexact matches, a measure of distance or similarity is required such as Levenstein Distance [51].

Many of the systems which perform the tasks described below reduce their task to that of score matching by transcribing performance or audio data to symbolic musical data.

Score-Performance Matching

Score-performance matching is the process of identifying correspondences between a musical score and a performed instance of that score at the event level. This detail of correspondence is essential in the analysis of performance issues such as the analysis of performance errors [103, 50], elements of musical expression (e.g. expressive tempo, timing, dynamics etc) [37], performance traditions [45], and the identification or analysis of a performer's individual style [112]. In these applications, it is necessary to analyse artefacts within the performance in the musical context in which they occur. For reasons to be discussed in Section 1.2 the performance information itself is not sufficient for these tasks, and analysis must be performed against the score. This type of correspondence can also provide a measure of distance or similarity between performed and symbolic music which is useful for identifying the score which corresponds to a particular performance (such as the 'query by humming' task which searches a musical database for a likely match to a user provided audio sample [57]).

In 1993, Large [50] presented an algorithm for matching scores and performances using dynamic programming. This was used to investigate cognitive planning by investigating the musical context of errors in piano performances [68]. Although Large briefly discusses polyphonic score matching, the matcher presented deals only with monophonic matching. Hoshishiba's system [40] included post-processing of results to improve reliability in the context of expressive music including ornamentation (performed notes which do not appear

individually in the score, but each correspond to the same notated event) and extreme expressive timing. Bolton's score-performance matcher [8] also developed a matching algorithm which was designed for the analysis of musical gesture and the adequate processing of ornamentation.

Dannenberg presented a matcher based on Dynamic Programming (DP) capable of matching polyphonic scores and performances (though the system was applied to score following rather than matching). Several algorithms perform score performance matching using Hidden Markov Models (HMM) (discussed in Section 3.1.2). The first systems matched a monophonic score to raw audio signals [76] and to frames of feature vectors derived from the raw audio signal [12].

Score Performance Alignment

Score following/Score-Performance alignment aims to map the relative time of a musical score directly to the absolute time of a performed instance of the score. In score following applications, this process must be performed in real-time, whilst a musical score is being performed. This places restrictions on the operation of such algorithms. In addition to estimating correlations between the absolute timeline of the performance and the relative time of the score, it must also decide when to report correspondences. Obviously a real-time algorithm does not have information on future performance events, so the algorithm should be able to elegantly adjust its calculated position when performance information causes a significant change in the current position.

Real-time score following can be used to perform tasks such as page turning, automatic computer accompaniment systems (of which there are legion) or synchronising time-based media. Off-line score performance alignment can be used for content-based indexing of audio information and synchronising time-based media.

Score following was pioneered by Vercoe [104] and Puckette [74]. In 1997, Grubb [30] presented a stochastic method which modelled the performance position as a probability distribution function based on the performer's source position (the last note performed), destination position (expected performed note), the estimated distance (relative time since source position) and the current observation (the current performed note). Later stochastic methods used Hidden Markov Models (HMM) to follow a score using either MIDI performance data [21] or audio features [66]. Dannenberg applied Dynamic Programming (DP) to score following in 1984 [17], and developed a polyphonic, DP based score follower in 1985 [6] (Dynamic Programming will be discussed further in Sections

3.1.2 & 3.2.1). Several algorithms use DP techniques to align polyphonic scores to audio features. Dannenberg developed a system which matched scores to audio [42] by rendering a score to audio using a synthesiser. Matching was then performed on the two audio streams. A modified version of this algorithm was applied to the query-by-humming task [60].

Performance alignment

Performance alignment attempts to align relative times between two different performances of the same piece. This can be used for content-based indexing of audio files [113]. Dannenberg [42] presented a system which is used for score-performance alignment, but the alignment is achieved by synthesising an audio representation of the score and performing audio-to-audio alignment with the performance. Bora [102] developed the only system known to the author which is designed for performance to performance *matching* (event-level correspondence between performance data). It analyses a performance through comparison with an 'expert' performance, examining differences in tempo, dynamics etc. It is intended for use as a training aid where students can gain objective feedback on the differences between their performance and the performance of the teacher. However, in theory, Dannenberg's score-performance matcher which renders scores to audio may be suited to performance alignment [42, 60] (though not at the event-level).

3.1.2 Underlying Technologies

Many algorithms have been developed for matching and aligning musical objects. The methodologies broadly fall into three categories according to the techniques used to find correspondence: Lookahead, Hidden Markov Models and Dynamic Programming (sometimes referred to as Dynamic Time Warping). Each of these techniques will be briefly examined in the context of matching or aligning scores and performances.

Lookahead

The simplest matchers, conceptually, use a deterministic, 'lookahead' approach [102, 73, 32, 37]

These algorithms iterate through the reference and subject, analysing at most a small window of each sequence. These matchers have a 'current position' which indicates a 'correct' correspondence between the reference and subject sequences. Successive correspondences are chosen by analysing the next item in

the subject against unmatched elements in the reference sequence. In real-time score following applications, the limited region of analysis is itself not an issue as future events are unknown. However, using a limited region of analysis (rather than considering all future correspondences or correspondence paths) means that these methods are unable to change previous correspondences. Because of this, once a series of incorrect correspondences have been made it is possible for these algorithms to lose track of the performance irreversibly. This 'pathological error' is caused when errors in the performance result in a performance position which is beyond the window of analysis. This can happen quite easily in passages which feature many repeated notes, or groups of notes. For this reason, Puckette [74] recommends that someone should be ready to intervene in case the real-time score follower loses synchronisation with the performance.

Hidden Markov Models

A Markov Model is a stochastic, finite state automaton. A Markov model consists of a set of finite states, transition probabilities (which describe the probability of changing from one state to another), a sequence of finite observable emissions, and emission probabilities (which describe the probability of observing an emission given the current state).

A Hidden Markov Model is so called because it has an unknown state sequence, but has known state transition probabilities, observable outputs, and emission probabilities. Thus, the challenge is to derive the most probable state sequence which could emit the observed output.

In the case of score matching, the performance data provides the observable output emitted by the unknown state sequence, and this state sequence represents the sequence of score notes which most likely correspond to each part of the performance. The probabilities of the model are often derived through unsupervised training on a set of data with known observable output and state sequences. Transition probabilities can be derived straight from the score [69], while emission probabilities are usually derived through an iterative training process. While this training process can be seen as a disadvantage, it does provide the opportunity to tune individual models towards particular instruments.

In 1999 two HMM based matchers were presented. Cano et al [12] used a vector of six features extracted from the acoustic signal as observable emissions (energy, delta energy, zero crossing, fundamental frequency, delta fundamental frequency and fundamental frequency error) in a monophonic score-performance matcher. Raphael [76] also presented a monophonic matcher which used analysis of acoustic data. This matcher was later used in the development of a polyphonic

score follower [66]. This score follower used a two-layer HMM model in which a higher state sequence modelled score events such as notes, rests, trills etc. and possible performance errors. This model was adopted in an algorithm by Schwarz et al [21] in which MIDI data was used as input rather than acoustic data.

Dynamic Time Warping

Dynamic Time Warping (DTW) is a method which allows the optimal correspondence between two sequences to be calculated. It provides an efficient method for the identification of temporal correspondence between two sequences and quantifies the similarity or difference which exists between the two sequences. In the context of score matching this term has been used interchangeably with Dynamic Programming (DP) [48] which is a method of solving problems which exhibit optimal substructure; i.e. the solution to the global problem can be derived from the solutions of the individual sub-problems. DP provides an efficient algorithm for performing DTW.

DTW can be applied to both symbolic sequences and sequences of audio features. It was first used in the musical context by Dannenberg [17]. Dannenberg's system performed real-time score-performance following by matching a monophonic score to symbolic performance data. Dannenberg later discussed two possible methods of using DTW to match polyphonic scores [6], though only one of these methods was implemented. It used a grid of values just like the previous DTW method, but the score was analysed as a sequence of note clusters (notes which share a common start time) while the performance remained a sequence of individual events. Thus each object in the score sequence could be matched to several notes in the performance sequence. Each value in the grid contained not only the optimal correspondence of the subsequence, but also a list of which score notes in the corresponding cluster have already been matched to performance notes.

Dannenberg's second method grouped both score and performance events into clusters. This allowed traditional DP algorithms to be used to match individual score and performance clusters to each other. However, this system was not actually implemented until Hoshishiba presented a polyphonic matching algorithm in 1996 [101].

DTW has also been used to perform alignment between audio data. Dannenberg [42, 60] performed DTW on a 'chromagram' representation of audio information. A chromagram is a sequence of vectors, each of which represents the spectral energy corresponding to each of the 12 equally-tempered intervals

C C#, D .... The score is rendered to audio using a MIDI synthesiser which is also converted into a chromagram. Thus the alignment of score and audio is transformed into an audio-audio alignment task. Dixon [113] presented a similar system which matches polyphonic audio using a DTW comparison based on Fourier analyses.

3.2 Using Dynamic Programming to Match Scores and Performances

The remainder of this chapter will focus solely on the matching of score and performance data using dynamic programming (DP) techniques. The different approaches and the effect these approaches have in the context of score-performance matching will be discussed. The main flaws in DP matching will be examined, and novel solutions to these problems will be presented and tested on real performance data.

3.2.1 The DP Method

In the symbolic comparison of musical scores or performances using DP, the score and performance is reduced to a sequence of ordered pitches. Thus all other information such as onset, duration, dynamics etc are discarded. The consequences of ignoring temporal information will be discussed in section 3.2.3, and possible solutions discussed in Section 3.4.

As explained in Section 3.1.2, DP is a method which finds the optimal solution to a problem by breaking it up into a series of subproblems. As opposed to the 'brute force' method which analyses every possible combination of global correspondence individually, DP methods find the optimal match for a pair of sequences by analysing the optimal match of subsequences. A *correspondence pair* contains an element from each sequence (in this instance, a note from the score and a note from the performance). All *possible* correspondence pairs are represented in a grid where each column represents a note in the sequence of score notes, and each row represents a note in the performance sequence. The grid is populated using an iterative algorithm which assigns each cell a 'score' or 'cost'. This value represents the quality of the *optimal* subsequence from that correspondence pair to the correspondence pair at the end of the grid. The global optimal correspondence is found by finding a path through this grid based on these values. The method is is described step-by-step below.

1. The score is represented as a sequence of pitches: $S(x)$ with length x_{max}.

The performance is represented as a sequence of pitches: $P(y)$ of length y_{max}.

2. A grid of size $x_{max} \times y_{max}$ is created where the position $G(x, y)$ represents the possible correspondence pair of $S(x)$ and $P(y)$.

3. For every correspondence pair, there is an optimal 'path' or sequence of correspondence pairs from $G(x, y)$ to $G(x_{max}, y_{max})$ which will identify the optimal match between the two subsequences $S(x \rightarrow x_{max})$ and $P(y \rightarrow p_{max})$.

4. From the position $G(x, y)$ there are 4 possible operations:

   (a) A correct match

   (b) A deletion (a note in the score which is missing from the performance)

   (c) An insertion (an extra note in the performance which does not correspond to a note in the score

   (d) A wrong match (the performance note corresponds to a note which is of a different pitch)

5. Correct or wrong outcomes result in a movement in the path to the correspondence pair $G(x + 1, y + 1)$.

6. A missing outcome leads to the pair $G(x + 1, y)$.

7. An extra outcome leads to the pair $G(x, y + 1)$.

8. Thus the optimal movement at correspondence pair $G(x, y)$ can be derived by evaluating the relationship between $S(x)$ and $P(y)$. and the optimal correspondence of the possible sub-sequences:

   - $G(x + 1, y) \rightarrow G(x_{max}, y_{max})$
   - $G(x, y + 1) \rightarrow G(x_{max}, y_{max})$
   - $G(x + 1, y + 1) \rightarrow G(x_{max}, y_{max})$

9. The quality of the optimal subsequence from $G(x, y)$ to $G(x_{max}, y_{max})$ is derived from the optimal movement and the quality of the grid position to which the movement directs.

10. The grid is populated iteratively with these values.

11. The global path is calculated by traversing the grid based on the values based on the movements described above.

There are two methods for measuring the quality of the correspondence between two sequences: cost and similarity.

1. The cost method attempts to minimise the difference between the two sequences and assigns a cost for each of the three error conditions: insertion ($cost_e$), deletion ($cost_m$), substitution ($cost_{xy}$). In the monophonic examples investigated in this theses, the value of $cost_{xy}$ is 0 when the pairs match and 1 when the pairs do not match.

2. The similarity method attempts to maximise the similarity between two sequences by assigning a value of similarity for each possible match pair (sim). In the monophonic examples investigated in this thesis, the similarity value (sim) is 1 when the two pairs are equal and 0 when they differ.

The optimal correspondence sequence is therefore one which accumulates the smallest cost or the maximum score in the shortest path. Using rule number 8 above, the grid can be populated iteratively using the following algorithms for similarity fig. 3.1(a) and cost fig. 3.1(b).

$$G(x,y) = \max \begin{cases} G(x+1,y+1) + sim \\ G(x,y+1) \\ G(x+1,y) \end{cases}$$

(a) Similarity-based method.

$$G(x,y) = \min \begin{cases} G(x+1,y+1) + cost_{xy} \\ G(x,y+1) + cost_m \\ G(x+1,y) + cost_e \end{cases}$$

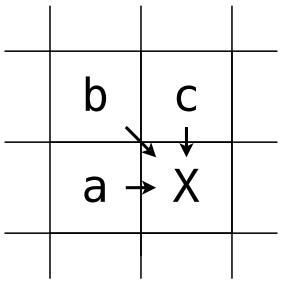
(b) Cost-based method.

Figure 3.1: Methods of populating a grid of correspondence pairs

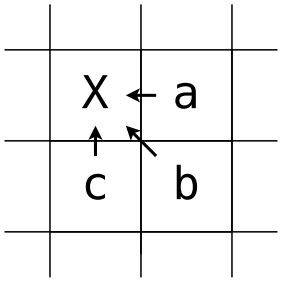
The cost algorithm assigns a cost to each transition which results in an incorrect correspondence. In this example, each error carries equal cost, however, in other implementations this may not be the case. Thus each cell represents the minimum possible error in an analysis of the subsequence from this cell onwards.

The similarity algorithm assigns a score to each transition resulting in a correct match. Each correct correspondence increments the value of the previous cell, thus each cell contains a value of the greatest number of possible correspondences. The rules in Figure 3.3 are then used to choose a path through the grid which corresponds to the optimum sequence of correspondence pairs. The

rules are based on the values of 3 cells surrounding the cell being evaluated. Figure 3.2 illustrates the location of the cells when populating a grid in reverse. The cell containing 'X' represents the cell being evaluated.



(a) Location of cells when populating a grid forwards.



(b) Location of cells when populating a grid in reverse.

Figure 3.2: Explanation of the locations of cells as described in the rules for grid population in figure 3.3

The following section provides a demonstration of the grids used in evaluating correspondences. It also explains the notations used in grids and sequences which describe correspondences. Figures 3.6 & 3.7 show grids populated using the similarity and cost methods respectively.

3.2.2 Explanation of graphs and notations used in the description of correspondences

The matching process is evaluated by inspecting the grids used in the DP process. Figures 3.6 & 3.7 show grids populated using the similarity and cost methods respectively.

The arrows through the grid indicate the optimum path through each grid and therefore the final correspondence. The cells which are traversed in the

$P(x) = S(y)$	Correct
$(b < a) \wedge (b < c)$	Wrong
$(a < b) \wedge (a < c)$	Missing
$(c < a) \wedge (c < b)$	Extra
$else$	Wrong

(a) Similarity-based rules

$P(x) = S(y)$	Correct
$(a = b) \wedge (a = c)$	Wrong
$(a > b) \wedge (a > c)$	Missing
$(c < a) \wedge (c < b)$	Extra

(b) Cost-based rules

Figure 3.3: Rules for traversing the optimal path through the grid. The location of cells (a,b&c) is described in figure 3.2

optimal path are coloured according to the movement, and thus the type of correspondence which is associated with the correspondence pair.

- Where a cell represents a correct correspondence, the cell value is coloured green. This represents a diagonal movement through the grid where the correspondence pair are equal.
- A red cell represents an incorrect correspondence. This corresponds to a diagonal grid movement when the correspondence pair are unequal.
- A blue cell represents either an insertion or a deletion.
  - A horizontal movement advances one element in the score without advancing through the performance. Therefore this represents a score note which is missing from the performance
  - A vertical movement advances one element in the performance without advancing through the score. This represents an extra note in the performance.

A correspondence sequence can be described using a sequence of score and performance values with symbols to represent incorrect correspondences. Figure 3.4 describes the notation used in the description of correspondence sequences. The sequence in figure 3.5 represents the correspondence in the grids in figures 3.6 & 3.7. This correspondence contains 3 errors: the B in the score is missing from the performance, the D is mistaken for another note in the performance, and an extra F has been added to the performance.

Symbol	*Significance*
$\oplus$	Extra note in performance
$\ominus$	Note missing from performance
$\otimes$	Substitution, score note is matched to a wrong note in the performance

Figure 3.4: Legend describing symbol used in sequences which represent score-performance correspondences

Score :	*A*	*B*	*C*	*D*	*E*	*F*	$\oplus$	*G*
Performance :	*A*	$\ominus$	*C*	$\otimes$	*E*	*F*	*F*	*G*

Figure 3.5: Sequence describing correspondences in 3.6 & 3.7

S\P	A	B	C	D	E	F	G
A	5	4	4	3	3	2	6
C	4	4	4	3	3	2	1
X	3	3	3	3	3	2	1
E	3	3	3	3	3	2	1
F	2	2	2	2	2	2	1
F	2	2	2	2	2	2	1
G	1	1	1	1	1	1	1

Figure 3.6: DTW grid populated and evaluated using the similarity algorithms.

S\P	A	B	C	D	E	F	G
A	3	3	3	4	4	5	6
C	4	3	2	3	3	4	5
X	5	4	3	2	2	3	4
E	5	4	3	2	1	2	3
F	5	4	3	2	1	1	2
F	5	4	3	2	1	0	1
G	6	5	4	3	2	1	0

Figure 3.7: DTW grid populated and evaluated using the cost algorithms.

3.2.3 Ambiguities in the output of DTW

There are certain situation in which the DP method is incapable of determining the exact combination of errors accurately. These ambiguous situations can all be traced to the reduction of temporal information to a sequence of ordered events. Without this information, a matcher must make an arbitrary decision on how to evaluate these situations. The rules and direction of path evaluation will determine the output of the matcher, and the output of the matcher will at least, be consistent. However, as will be discussed throughout the chapter, in certain cases it is desirable for a matcher to evaluate these in a particular way to suit the particular implementation.

$$
\begin{array}{ll}
Score: & A \quad B \quad B \quad C \\
Performance: & A \quad B \quad \ominus \quad C \\
Score: & A \quad B \quad \oplus \quad C \\
Performance: & A \quad B \quad B \quad C
\end{array}
$$

Figure 3.8: Repeated notes causing unreliable correspondences in DP matching

In the first example in figure 3.8, two consecutive score notes have the same pitch. Only one of these notes appears in the performance. Without temporal information such as onset time or duration, it is impossible to discern to which score note (if any) this performance note corresponds. Without this information,

the correspondence made by a DP matcher will depend on the priority of rules within the path evaluation algorithm, and the direction in which the grid is populated (and therefore the path is evaluated). The reverse applies to the second example above where one score note may match any of two performance notes.

$$\begin{array}{lcccc} Score: & A & \oplus & B & C \\ Performance & A & B & \otimes & C \end{array}$$

(a) Sequence representing correct correspondence

$$\begin{array}{lcccc} Score: & A & B & \oplus & C \\ Performance: & A & B & D & C \end{array}$$

(b) Correspondence as evaluated by DP matching

Figure 3.9: Technical error resulting in incorrect evaluation

Figure 3.9 describes a technical error followed by a substitution. The performer accidentally played a B at the wrong time, and played a D in place of the B in the score). The DP method will always create a correspondence between the B in the score and the B in the performance because this match will optimise the number of matches and the edit distance. Causing the correspondence shown below.

However, by using temporal information, it is possible to determine that the performed B cannot be matched to the score note because either the duration is too short, or, more likely, the onset time is incorrect. The onset time of the performed note D should show that there is a correspondence between this performed note and the B in the score.

The third example, figure 3.10, demonstrates a pair of sequences which has three possible interpretations:

1. The performance contains two incorrect notes (The notes have been played in the wrong order).

2. The performance contains one insertion and one deletion, the B is performed correctly.

3. The performance contains one insertion and one deletion, the C is performed correctly.

Each of these outcomes is possible, and again, temporal information is required to determine which possible outcome is correct in each particular in-

$$
\begin{array}{lcccc}
\textit{Score}: & A & B & C & D \\
\textit{Performance}: & A & C & B & D
\end{array}
$$

1.
$$
\begin{array}{cccc}
A & B & C & D \\
A & \otimes & \otimes & D
\end{array}
$$

2.
$$
\begin{array}{ccccc}
A & \oplus & B & C & D \\
A & C & B & \ominus & D
\end{array}
$$

3.
$$
\begin{array}{ccccc}
A & B & C & \oplus & D \\
A & \ominus & C & B & D
\end{array}
$$

Figure 3.10: Example of ambiguous situations which cannot be resolved using DP-based matching.

stance. Which option a matcher will choose depends upon the path evaluation algorithm (the direction of evaluation, and the rules for grid traversal, discussed in Section 3.2.5), and whether the population and evaluation algorithms are based on similarity or cost (discussed in Section 3.2.4).

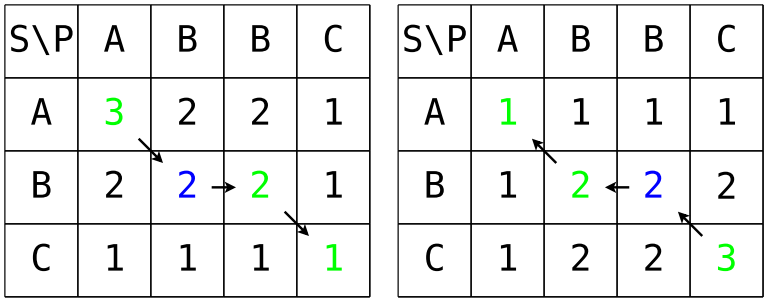
The final interpretation is only applicable in the context of polyphonic matching. In polyphonic music, it is possible that these notes should both be matched to the correct notes in the score. For example, in a piano performance with complex rhythm it is highly possible that two notes which appear in different hands may occur out of sequence in the performance due to extreme expressive timing. DP matching is incapable of creating out-of-sequence correspondences, and will always choose one of the aforementioned interpretations.

3.2.4 Cost vs Similarity

The first solution to the problem presented in Figure 3.10 is a result of giving greater priority to the optimisation of edit distance than is given to the optimisation of the number of matches. When priority is given to maximising the number of matches, the outcome of the matching process will be either the second or third interpretation. Bolton [8] believes that since it is assumed that the performer attempts to remain faithful to the score, it is correct to identify errors in a fashion which gives 'benefit of doubt' to the performer i.e. the method places a priority on maximising the number of correct matches rather than minimising errors. Perhaps this decision should be influenced by the quality of the performer. Neither method will prove accurate in all cases.

3.2.5 Direction of path evaluation

Thus far, all examples of DP have involved evaluating the path from $G(0,0)$ to $G(x_{max}, y_{max})$ and populating grid in reverse. It is equally possible to populate the grid forwards, and evaluate the path in reverse. A forward grid population routine makes it possible to extend the grid as new elements are added. If a new note is appended to the performance, only one more row of values needs to be calculated. Whereas, if the grid is populated in reverse, the whole grid must be recalculated whenever a new note is appended. Forward population has obvious benefits in the application of DTW to real-time score-following where the dimensions of the grid will change with each newly performed note.



(a) Forward path evaluation (b) Reverse path evaluation

Figure 3.11: Effect of forward and reverse path evaluation using the same rules

The direction of path evaluation also has a consequence on the output of the matcher. Using the same rules for population and grid traversal, forward and backward path evaluation will provide different correspondences in ambiguous situations. Figure 3.11 shows the grid and optimum paths for the same sequences using forward and reverse path evaluation. Although the path itself looks identical, the correspondences produced are different (a correct correspondence is accompanied by a diagonal movement *from* the corresponding grid position). In ambiguous situations such as these, there will always be a priority towards either later or sooner matches, and this will depend on the rules of evaluation, and the direction of evaluation (ambiguous situations are discussed further in section 3.2.3). It is impossible to derive the correct interpretation with the information presented to us. However, the default priority must be taken into consideration in certain applications. When the score and performance are analysed incrementally [113, 22], these situations are commonplace (for example in any repeated sequence of notes).

$$\begin{array}{lcccccc} Score: & A & B & C & A & B & C \\ Performance: & \ominus & \ominus & \ominus & A & B & C \end{array}$$

The performance sub-sequence above describes such a situation. The application may be real-time score following in which the performer has yet to perform the final three notes within the subsequence. However, if the default behaviour is to give priority to later matches, the accurate correspondence will only be indicated once the performer performs the second A. Therefore in applications such as these, the default behaviour should be to match the earliest possible correspondences which will guarantee an accurate correspondence as long as the performance remains accurate. Inaccurate correspondences can only be caused by an error in the performance.

$$\begin{array}{lcccccc} Score: & A & B & C & D & B & C \\ Performance & \ominus & \ominus & \ominus & D & B & C \end{array}$$

The future match error (above) is a similar situation, however, this is caused by an error in the performance. In the example below, a substitution in the first note of the performance has resulted in a subsequence which correctly matches a future subsequence. The subsequent performance note 'D' will correct the correspondence (below).

$$\begin{array}{lcccccc} Score: & A & B & C & D & B & C \\ Performance: & \otimes & B & C & D & \ominus & \ominus \end{array}$$

Dannenberg [16] describes this as 'strange behaviour' and overcomes this by analysing only part of the score. Analysis is performed using a window centred around the last match. However, this only prevents a future match occurring with a match which is outside the window. The reliability of this method will depend on the size of the window analysed. Too large a window may result in future match errors of the kind described above, while a window size which is too small may lose the performance position, and as long as this method maintains a fixed window size, it is possible that a combination of errors may cause a pathological error. Bolton [8] implemented real-time score following using forward path estimation and the entire score. Bolton's system introduced an extra grid which when superimposed upon the original grid, augmented the scores of particular correspondence pairs which in which the score and performance notes were equal in pitch. With a few extra rules in the path evaluation, Bolton was able to reduce the occurrence of future match errors dramatically without limiting the analysis to part of the score.

3.2.6 Efficiency

Despite the significant increase in efficiency the DP method offers over a 'brute force' comparison of all possible correspondences between two scores, there is still a lot of redundancy in the DP method. Most of the calculations in the grid are unused in the evaluation of the grid path (except in the accumulation of the scores/costs for other cells). Section 3.4 discusses ways to reduce the time and space requirements of the algorithm.

One method proposed by Chiba [87] imposes limits on the amount of the grid which should be populated. Assuming that the ideal path will follow the diagonal where x = y, if a limit was set on the maximum deviation from this ideal, it is possible to compute only a narrow band of cell values along this diagonal. However, this method relies on the fact that the correct path remains within this narrow band, otherwise a pathological error may occur. Therefore, the width of this band must be chosen carefully to compromise between increased time and space during calculation and accuracy of results. Dixon [113] improved on this method by using an evaluation of the grid at each iteration of population to determine the best direction of expansion. Therefore they were able to reduce the width of the computed band by calculating a band which followed the best path 'so far'.

3.2.7 Summary

This section has described how Dynamic Programming can be applied to score-performance matching. This is the first explanation which fully evaluates the shortcomings inherent in the DP method, the subtle differences in implementation (such as the direction of grid population or the use of cost/similarity based algorithms) & explained how these differences affect the final evaluation. This information is essential to create a matching algorithm which will evaluate the sequences musically, rather than assuming that the most accurate match is one which maximises correct correspondences or minimises errors. It is also essential to understand how different matchers will evaluate sequences at the micro-level, allowing a more profound comparison than a comparison of 'correct' correspondences. The next section will continue this analysis in the context of polyphonic matching.

3.3 Polyphonic Matching using DP

So far, only monophonic DP matching has been described. The difficulty in transferring DP matching into the polyphonic domain is that notes in the score which share a common start time can appear in any order in the performance. The DP method requires monotonicity, and thus is unable to match events which appear out of correct order.

This problem has been addressed in two ways. Dannenberg [6] proposed a method which compares clusters in the score against individual note events using DP. A grid is populated as before, but a far more complex algorithm is required for grid population. Each grid cell contains a list of 'used' pitches which represent pitches which have already been locally matched to the notes in the score cluster of the current correspondence. This prevents the same pitch in the score being matched to two equal pitches in the performance. Dannenberg reports that this method was successful in all tests, but stated that a *formal* proof of its optimality could not be provided. However, even if the algorithm could be proven to provide the optimum correspondence in all cases, Dannenberg's method still compares the score and performance as an ordered sequence of pitches. Therefore the method still suffers from the ambiguities which are inherent in a comparison which does not use temporal information. For example, given the following comparison, Dannenberg's method is unable to determine whether the 'correct' match is the chord CEG, EGC, or any partial combination of these notes. It can only provide the optimal match given a certain criteria.

$$\begin{array}{ll} Score: & C \quad E \quad G \\ Performance: & C \quad E \quad G \quad C \end{array}$$

3.3.1 Clustering

The alternative method to Dannenberg's is to represent both the score and performance as note clusters. This requires two modifications to the standard DP method. The first is a pre-processing stage which will group the performance notes into clusters of 'simultaneous' notes based on their onset times. A widely used method for grouping performance notes is to analyse the inter onset interval (ioi). If the difference between a note onset and the onset of the previous note is below a threshold ϵ, the notes will be assigned to the same cluster. The next note will be judged against the current note with the same threshold. It follows then that the onset times of a cluster may span a greater amount of time than the threshold, as long as each ioi is within the threshold. Different values for this threshold have been used ranging from 75ms [88] to 100ms [67, 6].

Unfortunately, in an expressive performance it is not so easy to group notes in such a fashion. The choice of threshold will be affected by many factors, most of which will vary throughout the course of the piece. The tempo of a performance will change from piece to piece and performer to performer. Even within the same performance, the tempo of a piece may change dramatically. Rolled chords would require a high threshold for notes to be successfully grouped, but a passage with a high density of notes may require a lower threshold to distinguish each note as an individual event. Thus, choosing the correct threshold is at best extremely difficult and at worst impossible. Section 3.4 will describe a novel method developed by the author which employs a dynamic clustering method to cluster notes based on a partially matched performance.

3.3.2 Partial Matching

The second requirement for polyphonic matching is an algorithm for evaluating partial matching. It is no longer sufficient for a comparison of two events to return a binary choice. In the example below, parentheses are used to denote chords. Neither C nor CE are absolute matches for CEG. However, CE provides a closer match, and should be matched to the chord. Therefore a formula is required to calculate the distance between items so that the matcher can select correspondences based on a partial match. This section will present new algorithms for populating a polyphonic DP grid. The rules required to evaluate these grids will be discussed in the following section.

Score:	A	(CEG)	F	
Performance:	A	(EG)	(C)	F

Deciding which note clusters are closer in a musical sense is not a trivial matter. Consider the following clusters as compared to the cluster CEG:

1. (CE)
2. (CEGB)
3. (CEGBD)
4. (C)

Which of these clusters is 'musically' closer to CEG? Cluster 1 differs from the original only by one deletion. Cluster 2 differs by just one insertion. A comparison of the edit distance would show these clusters to be equidistant from the original CEG. However, musically the second cluster is closer because more notes are correctly matched. This correlates with the action of the matcher

at the event level where, the number of matches should be maximised, and the number of errors minimised.

A comparison between clusters 1 and 3 is more difficult. Cluster 1 is a closer match based on edit distance. However cluster 3 has a match for all the notes in the chord. In this example it is less clear which of the clusters is a closer match to the original. A thorough comparison of musical 'closeness' would require analysis of the errors in the context of the score. This may show that a particular error or combination of errors are more likely, For example CEF might be judged to be closer than CEB♭ because substituting F for G is a likely technical error (on a piano, for example), whereas substituting Bb for G is neither technically likely nor likely as a cognitive error because Bb is very distant from CEG both harmonically and visually on a score. This discussion does not even touch on the possible arguments related to the harmonic context. Without this analysis it is wrong to assume that an insertion and a deletion can be reduced to a substitution. Therefore in polyphonic matching is concerned, it is wrong to create correspondences which represent substitutions.

The same is true of certain situations in monophonic matching (where at least one substitution and at least one insertion or deletion occur consecutively). In the analysis of monophonic correspondences, the use of substitution correspondences can simplify the visualisation and understanding of the matching process. However, applications should be aware that substitution correspondences may be arbitrarily assigned.

$$similarity = \frac{no\ of\ missing\ notes\ + no\ extra\ notes}{\max\left\{\begin{array}{c} S_{size} \\ P_{size} \end{array}\right\}}.$$

$$cost = 1 - \frac{no\ of\ missing\ notes\ + no\ extra\ notes}{\max\left\{\begin{array}{c} S_{size} \\ P_{size} \end{array}\right\}}.$$

Figure 3.12: Algorithms for comparing the similarity of two note clusters

All the systems compared in this thesis use the similarity algorithm in Figure 3.12. This algorithm returns a continuous value in the range 1 (complete match) to 0 (complete mis-match). The algorithm does not resolve additions and deletions into substitutions for the reasons given above. It will favour cluster 2 over cluster 1, and it will favour clusters 1 or 2 over cluster 3. In other words,

where the edit distance (insertions and deletions only) is equal, the algorithm will favour clusters which maximise matches.

In comparison, the algorithm proposed by Hoshishiba [40] compares only the number of insertions and deletions when comparing two clusters. Therefore, the system is unable to make the distinctions described above.

3.3.3 Polyphonic Path Evaluation

The introduction of the new comparison algorithms requires new rules for evaluating the optimal path through the grid. Figures 3.13 & 3.14 show two grids which have been populated using the score and cost methods respectively. The colour orange is now used to signify partial matches.

Score: A (CEG) D
Performance: A (CEGB) (CE) D

In the first grid in Figure 3.13 the second transition should be a diagonal movement which would represent a partial match between the second chords in both the score and performance. However, cell $G(x, y+1)$ has a higher value (and hence, according to monophonic matching, indicates a transition which will result in a more accurate match) than $G(x+1, y+1)$ which, using the previous rules would result in an initial transition to cell $G(x, y+1)$, followed by a match between (CE) and (CEFG).

S\P	A	CEG	D
A	2.75	1.75	1.00
CEGB	1.75	1.75	1.00
CE	1.67	1.67	1.00
D	1.00	1.00	1.00

Figure 3.13: Polyphonic DTW grid using similarity based algorithms. The grid shows that the path does not always follow the highest score.

The second grid, shown in Figure 3.14, is populated using the cost algorithms. This shows a similar situation when using the cost-based algorithms.

The second transition should be a partial match between the second chords in the score and performance. However, the path which has the lowest cost would match CEG in the score to C in the performance.

S\P	A	CEG	D
A	2.25	3.25	4.00
CEG	2.33	2.25	3.00
CE	1.67	1.33	2.00
C	1.67	0.67	1.00
D	2.00	1.00	0.00

Figure 3.14: Polyphonic DTW grid using cost based algorithms. The grid shows that the path does not always follow the lowest cost.

In both the cost and the score methods, this is a situation where the optimal path is not a transition to the cell with the highest score or lowest cost. To evaluate the path correctly, the quality of the current match must be considered in addition to the value in the possible destination cells. In the first grid, a movement to $G(2,2)$ reduces the score to 1.00, but a match of value 0.75 has been made in the process. A movement to $G(1,2)$ reduces the score to 1.33, but no match has been made in the transition. Thus, the introduction of partial matching requires the cost as well as the result of the transition to be assessed.

$P(x) = S(y)$	Correct		
$a = b = c$	Wrong/Partial	$P(x) = S(y)$	Correct
$a < b \wedge a < c$	Missing	$b < a \wedge b < c$	Wrong
$c < a \wedge c < b$	Extra	$a < b \wedge a < c$	Missing
$h > c > a = b$	Partial	$c < a \wedge c < b$	Extra
$h > a > c = b$	Partial	$h = b + cost$	Partial
$else$	Wrong		

Figure 3.15: New rules for traversing the grid using the similarity (a) and cost based (b) methods for polyphonic sequences.

The new rules shown in Figure 3.15 show modifications to the original rules for score and cost algorithms which will find the optimal path through a grid using the comparison function in Figure 3.12. There is however, one major change to the operation of the cost method. The first new rule added to the list of cost-based rules is the rule of first match. This means, that when evaluating a path, if a correspondence pair are equal, then they will be matched (irrespective of the score/cost values of neighbouring cells). The consequence is that when using a backwards path evaluation (and thus, forwards grid population) the algorithm will now give priority to later matches in ambiguous situations.

The same is true in the implementation described by Hoshishiba [101] who states that finding the optimal path through the grid is a question of 'simply finding the path which minimises the cost'. Figure 3.16 shows a grid populated using Hoshishiba's cost algorithm. The cost algorithm presented by Hoshishiba increments the cost based on the number of insertions and deletions. The correct path through this grid does not always depend solely on the cost values in the cells. The path which has the lowest cost in this instance would miss the correct correspondence of the C major triad, and match the triad in the score to CE in the performance. Therefore, this method must also use the rule of first match, and will thus give priority to later matches when used with a forward path evaluation.

S\P	A	CEG	D
A	0.00	3.00	4.00
CE	2.00	1.00	2.00
CEG	5.00	2.00	5.00
D	6.00	3.00	2.00

Figure 3.16: Grid populated with Hoshishiba's cost algorithm. The optimal path does not follow the lowest cost. Correctly matching the C major triad requires the rule of first match.

3.4 Improving the DP method

The operation and inherent deficiencies in DP based matching have been discussed in Sections 3.2.3 and 3.3.1. This section will present novel methods to improve the performance of score performance matching in these areas. One method increases efficiency and accuracy of the DP method by analysing the performance using smaller overlapping windows rather than one large grid. When populating each new grid the thresholds for clustering performance notes into chords will be adjusted according to the local properties of the score and the performance.

Hoshishiba [40] suggested a second stage to the DP matching process which attempts to correct errors in the DP evaluation by searching for more alignments based on temporal information. Improvements to this analysis stage are presented and tested which rely on analyses of the score and performance.

It will be shown that the method presented by Hoshishiba [40] is only capable of correcting certain shortcomings in the DP method. It will be demonstrated in theory and in tests that score-performance matching can be further improved by first removing false positive matches from the evaluation based on temporal analysis before searching for further alignments.

3.4.1 The Dynamic Matcher

As explained in Section 3.3.1, the process of grouping performance events into clusters of simultaneous objects is extremely error prone. The acceptable inter onset interval of performance notes which appear simultaneously in the score can vary dramatically not only between pieces and performers, but within an individual piece. An improved method, such as the one presented here would use local properties of the score and the performance to alter the clustering threshold dynamically. Information such as the instantaneous tempo, the relative inter note onset interval, the number of simultaneous notes in the score and score markings eg. arpeggio, rubato etc. can all be used to influence the clustering algorithm. For example, a passage within the score which contains shorter inter note intervals would require the threshold to be reduced to decrease the chance of grouping consecutive notes into the same cluster, whereas passages with larger inter note intervals will allow a wider threshold value. Where a section of a performance contains liberal use of spread chords, it would be necessary to increase the threshold to ensure that such chords are correctly grouped. However, the clues within the score with respect to timing require translation into absolute time which, in turn requires the instantaneous tempo

of the performance.

While it is possible to extract tempo and beat information directly from the performance data, a far more accurate analysis can be achieved using matched score and performance data. Tempo and beat analysis based solely on the interpretation of performance onset and durations (and possibly note dynamics or accent) to derive instantaneous tempo is likely to misinterpret the tempo by a factor of two (i.e. doubling or halving the predicted tempo). In contrast, a matched score and performance can relate the relative times of notes within the score to the actual performed times to ensure (as long as the score-performance is correct) that an accurate evaluation of the instantaneous tempo can be calculated.

The system presented in this thesis uses an iterative matching process. The score and performance are analysed using overlapping frames. The results of each frame are used to analyse the instantaneous tempo of the piece. This tempo, and an evaluation of the context within the score is used to dynamically adjust the clustering threshold intelligently. This threshold is then used to re-evaluate the performance notes into clusters for the following frame.

The iterative method used by Dixon [113] might at first seem suitable for this process. The forward grid population allows each successive iteration to be appended to the previous match value. The constraints set on the width of the path provide further improvements in efficiency. However, as explained in Section 3.3.3 a forward grid population results in priority being given to later matches. This type of error is unacceptable in this situation.

Consider the examples below. As the matcher progressively analyses the score and performance, the arbitrary boundaries have caused an element within one sequence to be included in the grid while committing its corresponding element in the opposite sequence. In each example, the matching algorithm has made false correspondences in situations where the performance is an accurate rendition of the notated music. While this error would be corrected when the analysis boundaries are extended, it can cause a significant error in the calculation of instantaneous tempo which will be reflected in the clustering threshold used in the next analysis frame.

$$\begin{array}{cccc} A & B & C & C \\ A & B & \ominus & C \end{array}$$

$$\begin{array}{cccc} A & B & \oplus & C \\ A & B & C & C \end{array}$$

In the first example, the inter onset interval between B and the first C in the

performance will be evaluated against the relative inter note interval between B and the second C. This will lead to a tempo estimation which is higher than the actual amount. Similarly, the situation in the second example will lead to an artificially low estimation of the instantaneous tempo.

In theory, this could be improved by analysing the score and performance to ensure that the window boundaries do no not occur in such a situation. However, calculating where this boundary should be is non-trivial, and in many cases futile e.g. where there are significant repeated notes or chords.

Therefore, the only way to employ a dynamic threshold is to use reverse grid population routine. This means a new grid must be calculated for each analysis frame. If the entire score and performance analysed thus far was to be included in each frame, the number of computations would rise dramatically. Thus, the score and performance are compared using overlapping windows as described below, and in Figure 3.17.

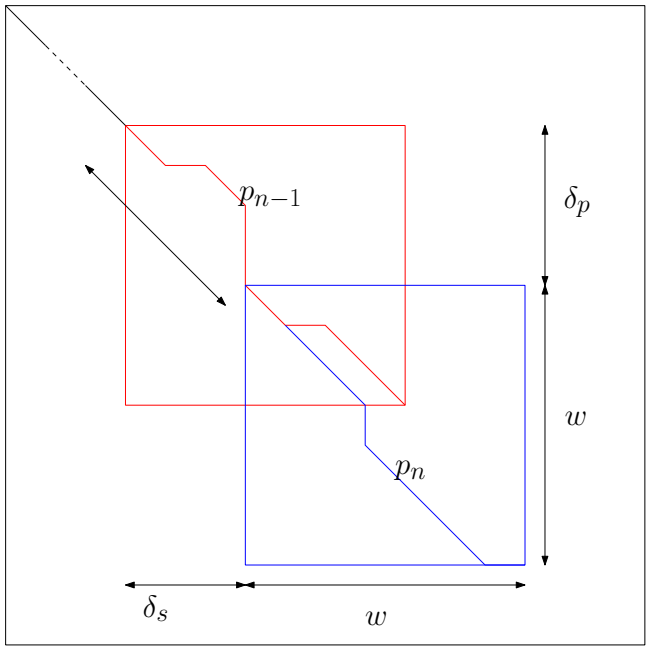


Figure 3.17: Evaluating performance using a sliding analysis window.

1. The first n score and performance clusters are obtained using an initial clustering threshold. This initial threshold may be based on information regarding the tempo indication on the score.

2. These clusters are used to populate a grid of size $n \times n$ using reverse grid

population and similarity-based algorithms.

3. The path through this grid is evaluated.

4. The correspondences obtained from this path are used to evaluate the tempo at the beginning of the next analysis window.

5. The score is analysed to extract information which may affect the threshold of future performance events eg. the minimum inter-onset interval, articulation and tempo markings. This information, along with the tempo analysis is used to determine a new clustering threshold.

6. A new analysis frame is created with these performance clusters and clusters from the score.

7. The δ points in the previous correspondence which relate to clusters not included in the new analysis frame are added to the list of final correspondences.

Each frame analyses a portion of the 'global grid' which is enclosed by the positions $G(m, n)$ and $G(m + \omega, n + \omega)$. The next frame begins at some offset from the original frame $G(m+x, n+y)$. The correspondences between the score and performance between the positions $G(m, n)$ and $G(m + x, n + y)$ are taken to be 'true' and are appended to the list of accurate matches. All other matches within a frame are discarded when a new frame is analysed.

The values of δ_s and δ_p which define the offset of the subsequent frame are derived from the analysis of the current frame. The start position of the next frame must coincide with a correct or partially correct correspondence. This ensures that the windows follow the correct path through the global grid. Otherwise, if the correct path was to lie outside the narrow band of analysis, the matcher would reach a pathological solution. Making windows begin on a correct correspondence also ensures that no errors are introduced as a result of the start boundaries. Such errors could cause incorrect correspondences to be included in the final list of correspondences, and cause inaccurate tempo calculations leading to incorrect clustering when initialising the next frame.

Notice that the path of the new grid (blue) does not exactly follow the remainder of the previous path (red). Errors are likely to occur due to the closing boundaries of the grid. This is unavoidable unless the grid is analysed in its entirety. This is why the windows must overlap rather than starting where the previous window finished. Errors induced by the position of the closing boundaries are unlikely to have any effect on the final output as long as there

is sufficient difference between the overlap amounts δ_s and δ_p and the overall window size ω.

This method is not immune to future match errors (discussed in Section 3.2.5). However, these will only occur when two criteria are met. Firstly, the future match error must be caused by a performance error which makes a subsequence exactly match a later subsequence. Secondly, the future subsequence must occur within the same analysis frame. The chances of such a situation are low, especially in a proficient performance. However, the chances are increased in passages which contain very closely repeated notes, chords or phrases.

The size of the window has an effect on reliability and efficiency. A larger window size leads to an increase in computation, and an increase in the possibility of future match errors. A smaller window size increases the chance of following a pathological path through the grid.

3.4.2 Interpolation

As described in section 3.2.3, the discarding of temporal information introduces inherent ambiguities which limit the accuracy of DP matching algorithms. However, once a score and performance have been matched, it is possible to perform further processing on the correspondence sequence using the previously discarded temporal information to identify and fix these errors when they occur. Hoshishiba previously used a spline interpolation to improve the output of a DP-based matching algorithm. This section will discuss the application of interpolation to DP matching and propose further improvements which make use of temporal information and information within the score.

Hoshishiba [101, 40] used DP matching to match scores to recorded MIDI data of piano performances. The DP matching was performed using cost-based algorithms which was augmented by a secondary analysis stage. The secondary stage attempts to identify correct correspondences which the standard DP method was unable to detect. Hoshishiba's algorithm does this by using spline interpolation to estimate the time at which any unmatched score notes *should* have been performed based on the existing correspondences. The performance is searched for any unmatched performance events which match the pitch of the score note, and lie within some threshold of the estimated onset time. A correct correspondence is created between the score note and the closest performance event, should one be found which matches these criteria.

This method is capable of correcting only two possible errors: out-of-sequence errors and errors caused by the incorrect clustering of notes. This is because the secondary analysis assumes that the output of the DP stage is correct. The

secondary stage is only capable of creating new correspondences which the DP stage did not create. The example discussed in Figure 3.3.2 represents a clustering error where the C in the performance has been played early, and the clustering algorithm has included the note in the chord. This situation can be resolved by using spline interpolation to create a new correspondence as long as the unmatched performance note occurs within the specified threshold of the estimated performance time.

Score:	A(CEG)F
Performance:	A(C)(EG)F

However, If we revisit the ambiguous situations discussed in Section 3.2.3, it is apparent that of these situations, only one particular situation can be corrected by the method described above. In the example below, two notes in the performance appear swapped in time. The DP matcher will choose one of the interpretations detailed below. However, in polyphonic music it is possible that the B and the C in the performance correctly correspond to the respective notes in the score. Expressive timing between separate voices, such as the two clefs within a piano performance, can lead to notes which are out of sequence in a strict sense of order, but are still within acceptable timing thresholds with respect to the rhythm of the voice in which they occur.

Score:	A	B	C	D
Performance:	A	C	B	D

A	B	C	D
A	$\otimes$	$\otimes$	D

A	$\oplus$	B	C	D
A	C	B	$\ominus$	D

A	B	C	$\oplus$	D
A	$\ominus$	C	B	D

The faults which are inherent in DP matching result in correspondences which are falsely reported as correct matches. Therefore to have any considerable improvement in the final, overall correspondence, the output of the DP matcher must first be analysed to identify and remove these false positive matches.

Removing false positive matches

Hoshishiba used spline interpolation can be used to estimate the time at which notes corresponding to unmatched score notes should occur in the performance. This can also be used to measure the consistency of existing matches in an evaluation. An algorithm was created which evaluates the consistency of existing matches based on the times of neighbouring correspondences.

The algorithm analyses the two correct correspondences immediately preceding the correspondence in question, and the two correct correspondences immediately following. Using these four relative/absolute time pairs a cubic spline is evaluated. From this spline, the proposed onset of the intermediate correspondence can be estimated. The consistency of the correspondence can be measured as the difference in time of the onset of the matched note(s) from the estimated onset time. If the difference between the estimated and performed time of a note are found to exceed a certain threshold, the match is removed from the evaluation.

A static threshold cannot take into account changes in the performance (changes in tempo) or changes in the score (a change in the density of notes). Therefore the threshold is based on an analysis of the performance and the score.

$$\epsilon_1 = t_{prev} + \xi(t_{est} - t_{prev})$$
$$\epsilon_2 = t_{est} + \xi(t_{next} - t_{est})$$

Figure 3.18: Calculation of dynamic thresholds

The calculation of dynamic thresholds is shown in Figure 3.18. The thresholds are based on a proportion ξ of the distance between the estimated performance time t_{est}, as calculated using the cubic spline, and the estimated performance time of the notes which immediately precede and follow the subject note (t_{prev} & t_{next}). The time of preceding and following notes is estimated based on the instantaneous tempo derived from the cubic spline, and the relative onset times extracted from the score.

Thus, a performance note is deemed to be within the thresholds if the following condition is true:

$$\epsilon_1 < t_{perf} < \epsilon_2$$

3.4.3 Interpolation Alignment

Hoshishiba used spline interpolation to improve upon the DP matching method. This method iterated through an evaluation searching for unmatched score notes. When an unmatched score note was found, the proposed onset time of this note in the performance was estimated using a cubic spline based on the relative and absolute times of correct correspondences surrounding the unmatched note. The performance was searched to find suitable notes to align to the unmatched score note. A performance note would be matched if it represented the same pitch and it's onset time lay within a certain static threshold and the onset time estimated using the cubic spline.

However, as discussed in section 3.4.2, this static threshold is incapable of accounting for changes in pitch or changes of the density of notes within the score. Therefore a new alignment algorithm was created based on the system used for removing false positive matches. This system creates thresholds dynamically using the same method as used in section 3.4.2. Although both the identification of false matches, and the creation of new matches use the same method of calculating thresholds, although the proportion ξ need not be the same.

It is also proposed that, in the case of polyrhythmic pieces such as Chopin opus 66, it is possible to perform the interpolation stages of each voice (or hand) individually. Thus reducing the effect out-of-order notes will have on the interpolation process.

3.5 How to evaluate matching algorithms

Evaluating the performance of a matching algorithm *thoroughly* is an extremely laborious task. Hoshishiba[40] assessed the performance of his algorithms by comparing the number of correct correspondences created by the matcher (100% correct matches representing the 'optimal' performance). However, this technique makes two assumptions. Firstly, it relies on the performance being an accurate representation of the score. In this sense, an *accurate* representation of the score means that for each note in the score there is exactly one corresponding performance note, and that there are no unnotated performance notes. Secondly, it assumes that the transcription of the performance has been performed with absolute accuracy. Neither of these assumptions can be made lightly. Basing performance purely on the basis of the number of correct correspondences makes no judgement of the *quality* of the matches. Furthermore, a perfect performance of a piece does not pose a rigorous test for a matching

algorithm, particularly its robustness against pathological errors or the correct identification of errors. The identification of errors is just as important as identifying correct correspondences. In the analysis of performance issues inaccurate matches (false positives) between notes can cause an equal disturbance in the results of analyses as missing correspondences (false negatives). In this context, an accurate evaluation of the performance would be the total number of false correspondences (false positives and false negatives).

3.5.1 Evaluation procedure

The performance of various matchers will be assessed in this chapter. These evaluations are performed using two types of data: artificially generated performance data and data from two real performances.

An 'authoritative' evaluation is required to be able to examine the correspondences evaluated by the matcher to ascertain whether they are accurate. The number of correspondences between notes in the score and the performance judged by the matcher to be correct is not an adequate assessment of the performance of a matcher. This does not assess whether the correspondences are 'accurate'. To assess this, the correspondences must be evaluated against an authoritative version which allows the following information to be calculated:

- TP(%): (True Positives) This is the number of correct correspondences the matcher accurately identified from the authoritative list.
- FN(%): (False Negatives) This is the number of accurate correspondences which the matcher failed to identify.
- FP(%): (False Positives) The number of correspondences inaccurately identified as correct.

When artificially generating performance data it is possible to generate an authoritative evaluation based on notes removed or added from a performance. However, it is possible that the random insertion and deletion of notes results in notes inserted at such a time and pitch that a correspondence between new notes and notes in the score is valid.

The authoritative versions of the real performances were evaluated by hand, using both the audio and the MIDI data collected from the performance. The performance of each matcher was tested against the authoritative list of accurate correspondences, providing an accurate analysis of how a matcher has performed. This is a very laborious task, therefore in this thesis only two real performances are evaluated.

Artificial Performances

The artificial performances are generated from the fifth prelude from Shostakovitch's *24 Preludes and Fugues*, Opus 87. A performance is rendered straight from the score with no expression or variation in pitch, time or tempo. From this 'vanilla' performance, different performance data was generated using three methods to transform the performance:

1. Applying a continuous change in tempo across the entire piece.
2. Spreading the onset times of notes within the same chord.
3. Inserting & removing notes from the performance.

To alter the tempo of the performance, the onset time of each note was adjusted using the equation in figure 3.19 where t is the original onset time and x is the factor used to apply a continual change in tempo.

$$t_{new} = t^x$$

Figure 3.19: Equation used to alter the tempo of a performance.

Chords are spread such that the onset time of notes in a chord are evenly spread throughout a timespan beginning from the original time of the chord. The timespan is equal for all chords throughout the piece.

The removal of notes is a random process. The sole condition is that only notes present in the vanilla performance may be removed from altered performances. The process An onset time which lies within the existing length of the performance is generated randomly. At this position a chord is inserted which contains the same pitches as the chord immediately following it in the score. Inserting notes in this fashion creates performances which are far more difficult to match successfully.

Using a combination of these methods, data can be generated which allows specific aspects matching algorithms to be examined in isolation and permits a more scientific approach to evaluation.

Real Performances

The real performances used in the tests in this study were recorded using both MIDI and audio data. MIDI gesture data was recorded using a 'Piano Bar' which is manufactured by Moog. The Piano Bar directs an infrared beam onto each key of the piano. Key presses and velocities are detected by analysing the angle of deflection of the beams. This allows gestural data to be recorded

without affecting the sound output of the piano, or the technical aspects of the performance.

The results of many of the matchers (results are compared in Sections 3.7.1 & 3.7.2) show that the number of correct correspondences a matcher has assigned is not an accurate measure of the overall accuracy of a matcher. In many cases, a matcher with a higher number of total correspondences has less accurate correspondences than other matchers with a smaller total number.

Ideally, the authoritative correspondences would be evaluated using an interface which simultaneously represents the score, performance and correspondences in a graphical way. However, the lack of an open API for the graphical rendering of musical notation makes this infeasible at the current time. The time required to evaluate each performance against the score and the time required to evaluate the output of each algorithm against the authoritative version has limited the number of performances on which the algorithms could be tested. To counterbalance the lack of material, pieces have been chosen which pose particular challenges for matching algorithms. Additionally, the performers recorded had limited exposure to the particular pieces, increasing the likelihood of performance errors.

The first piece is the fifth prelude from Shostakovitch's *24 Preludes and Fugues*, Opus 87. Although the piece is not particularly complex, it poses particular problems for matching algorithms. The presence of repeated chords which are identical or very similar increases the chance that the matcher will inaccurately evaluate a performance that contains performance errors. The particular performance chosen was played with a very romantic interpretation which makes considerable use of tempo variations and spread arpeggiated chords. The tempo variations and arpeggios mean that a single, static clustering threshold will not perform accurately across the whole piece.

Two errors are of particular interest in this performance. The first error occurs in bar 6. This bar is displayed as it appears in the score in Fig. 3.20(a) and a transcription of the performance in Fig. 3.20(b).

The section of the performance from the second beat to the end of the bar in the right hand matches exactly what is notated from the first beat until before the third beat. Although the timing information proves otherwise, a DP matching algorithm is likely to match all or part of the last two beats in the performance to the first two beats in the score. This is because an error in performance is, by coincidence, a suitable match for another part in the score.

A similar error has occurred in bar 56 (see 3.21). The performer has played something which is similar to the score, but the melody has been slightly trans-

formed and played later than notated. This has the potential to confuse the matching algorithm in the same way as the previous example, though to a lesser extent.

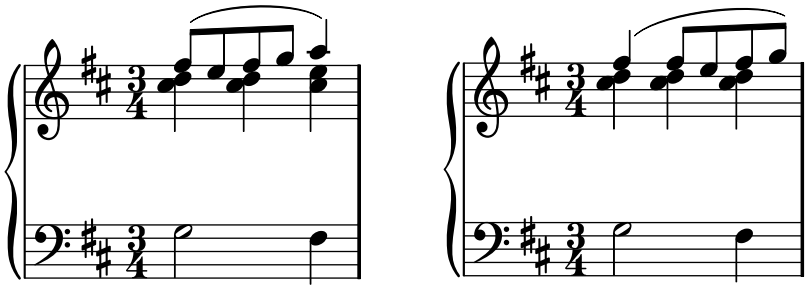
The second piece is the first section from Chopin's Fantasie Impromptu, Opus 66. This is one of Chopin's better known pieces, and features a cross rhythm throughout the first section with semiquavers in the right hand against sextuplets in the left hand. The rhythmic complexity, speed and the constantly changing figurations make it a challenging piece. It is a common characteristic of Chopin's works to have a steady left hand accompaniment against a free right hand melody. Expressive interpretations can lead to a significant number of situations where notes in the left and right hand appear in the wrong order with respect to the score, but in the correct order with respect to each hand individually. The polyphony in the first section rarely exceeds two simultaneous notes, though the complex and expressive timing will pose a challenge for clustering algorithms. However, performance matching will be dominated by the ability of the algorithm to cope with expressive timing, specifically the resolution of out-of-sequence notes. A performance was chosen which exhibited considerable variation in expressive timing between the left and right hands. The performance includes roughly 200 instances of out-of-sequence notes from a total of just under 1000 notated notes, and 150 instances of missing or extra notes. This performance should provide a thorough test of a score performance matcher's capabilities.

3.6 Benchmark Results

3.6.1 Static vs. Dynamic Matcher

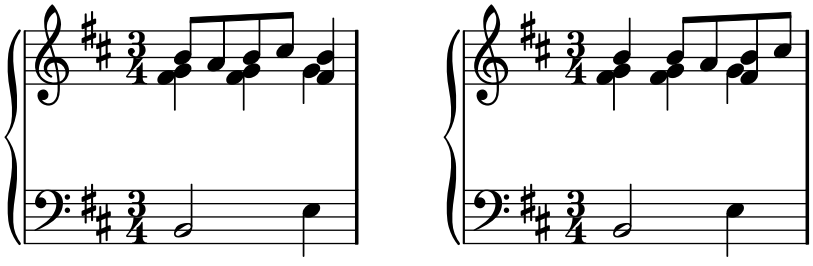
The dynamic matcher was tested against a static matcher which used identical DP rules (i.e. the grid population and path evaluation rules were exactly the same). Unlike the dynamic matcher, the static matcher used a single grid for evaluating the performance. The sliding window alone offers only an improvement in efficiency. In fact, in some cases the dynamic matcher was unable to follow the performance accurately enough to ensure that the analysis window followed the correct analysis path. This resulted in pathological errors from which the matcher was unable to recover.

In each of the experiments, the dynamic matcher was testing using different values of 'strictness'. The strictness determined what fraction of the estimated minimum inter onset interval was used to as the threshold. It was assumed (without any psychoacoustical experiments) that 0.5 might prove to be an op-



(a) Notated (b) Performed

Figure 3.20: Shostakovitch Prelude No. 5, Bar 6.



(a) Notated (b) Performed

Figure 3.21: Shostakovitch Prelude No. 5, Bar 56.

timal value for this. The static matcher was tested using a range of thresholds appropriate to the tempo of the piece.

The matchers were tested against artificial performances which were altered by spreading chords and altering the tempo of the performance. This tests the ability of the dynamic matcher to follow tempo changes in the performance and it's ability to dynamically adapt clustering thresholds accordingly.

All chords within the performance were spread by an amount of time equal to half the smallest inter-onset interval thus ensuring that chords do not overlap. From this, six version of the performance were created with varying tempo factors ranging from 1.0 (no change in tempo) to 1.5 (which resulted in a performance which was lengthend by a factor of 20).

In this test, a dynamic matcher with a clustering threshold of 0.5 was tested against static matchers with a range of clustering thresholds. The results of the evaluation can be seen in figure 3.25.

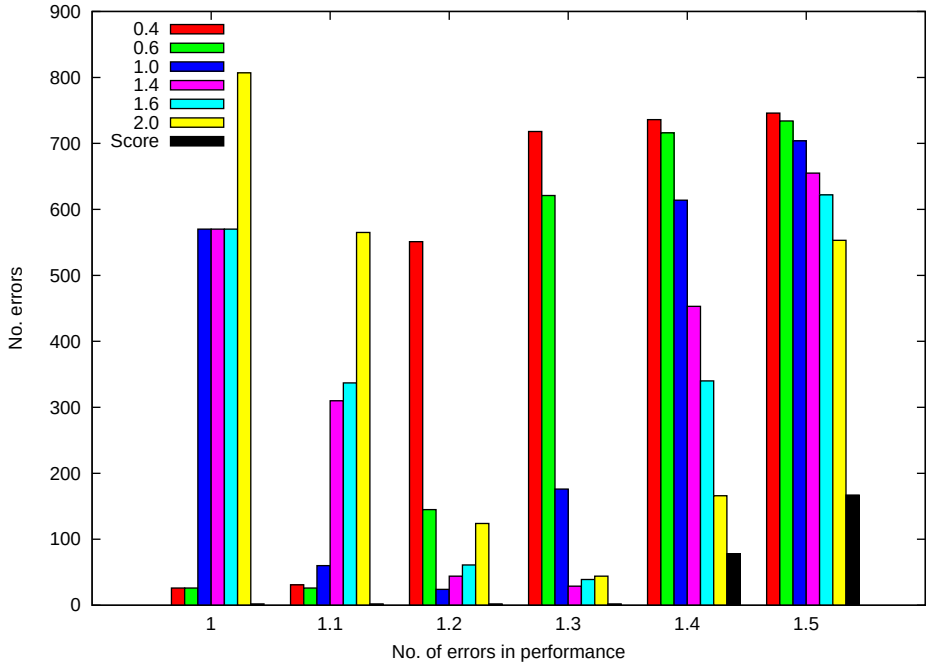


Figure 3.22: Comparison of the static and dynamic matchers using artificial performances.

In the first four performances, at least one static matcher performs reasonably well. However it can be seen that as the deceleration of the tempo of the performance increases, the optimal static threshold increases. The dynamic

matcher matches these 4 performances flawlessly. In the final two performances the deceleration of tempo becomes more severe and a static threshold is incapable of clustering notes into chords accurately throughout a large proportion of the piece. The dynamic matcher does not accurately identify all correspondences though still significantly outperforms the static matcher at all thresholds.

3.6.2 Removing false positives

Using artificial data, the effect of removing inconsistent correspondences before creating new correspondences was tested. The test performances contained introduced errors, but were of constant tempo, with no dispersion of notes within chords. Eight performances were created which contained between 50 and 400 introduced errors. This data was evaluated with two matchers using score-based interpolation alignment. One matcher removed inconsistent matches before aligning new matches, this will be referred to as the 'realigner', the matcher which only searched for new correpondences to align will be referred to as the 'aligner'. The unalignment stage was performed with a threshold of 0.5 and the alignment stages were tested at a range of thresholds in the range 0.1 to 1.0.

The realigner outpuerformed the aligner in most of the tests. Both matchers performed optimally using an alignment threshold of 0.2. Figure 3.24 shows the performance of these matchers at that threshold.

Figure 3.23 displays the performance of these matchers at varying alignment thresholds averaged over all performances. The performance of the matcher which did not remove inconsistent performances displayed little variation in accuracy across different alignment thresholds. This suggests that many of the errors in the DP matching process resulted in inaccurate correct correspondences rather than unaligned notes. Therefore there were fewer notes available to be aligned by the matcher. At higher alignment thresholds, the average performance of the unaligning matcher drops below that of the alignment-only matcher. In these situations correct correspondences which have been removed due to inconsistency are being matched to wrong clusters because the alignment threshold is too high.

3.6.3 Score Realignment vs Static Realignment

These experiments use artificial performances to test score-based interpolation alignment against alignment using a static threshold. The matcher using a static threshold is equivalent to the matching algorithm presented by Hoshishiba [40]. The test material consisted of eight performances without chord spreading but

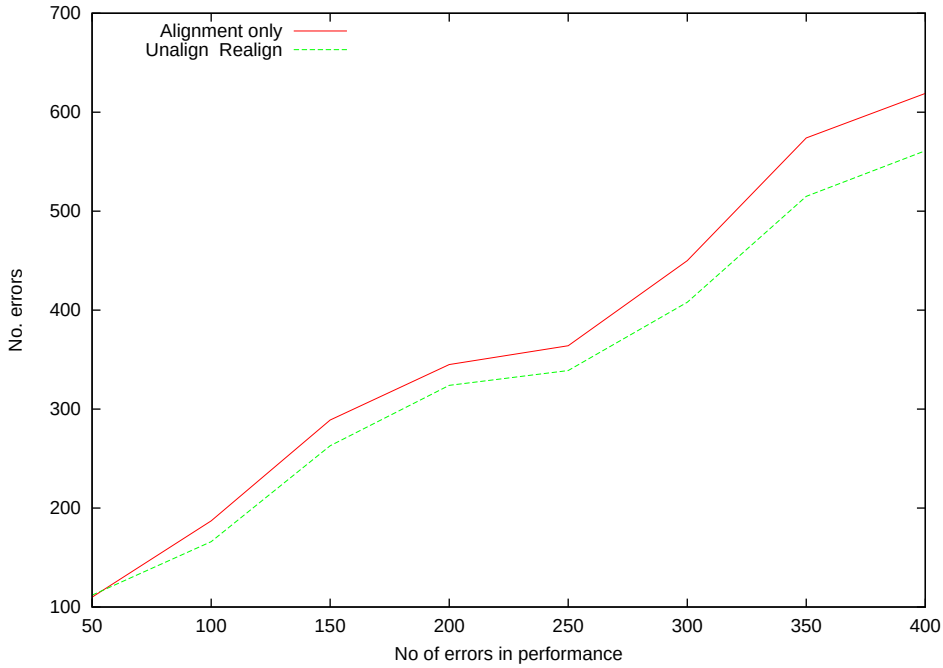


Figure 3.23: Average performance of interpolation alignment with and without unalignment stage.

with ramped tempo and errors introduced into the performances. Eight performances were used with error rates ranging from 50 to 400. The continuous change in tempo should demonstrate the advantage of using dynamic alignment threshold based on the instantaneous tempo of the performance.

Each performance was first matched using the dynamic matcher, then processed using interpolation based unalignment with a threshold of 0.5. The unalignment process ensures that more notes are available for realignment and as demonstrated in section 3.6.2, the unalignment process is likely to provide a more accurate final analysis. The static matcher was tested at a range of thresholds from 0.2 to 1.0. The score-based alignment process was tested at a threshold of 0.5.

Figure 3.25 displays the results for each matcher tested against each performance. Figure 3.26 contains the average performance of each matcher.In all cases the score-based realignment outperforms the static realignment method.

As the tests on the static matcher (figure 3.25 the static threshold, the optimal static threshold can be seen to change as the tempo of the piece changes. However, as the variation in tempo increases, there is less distinction between various static thresholds.

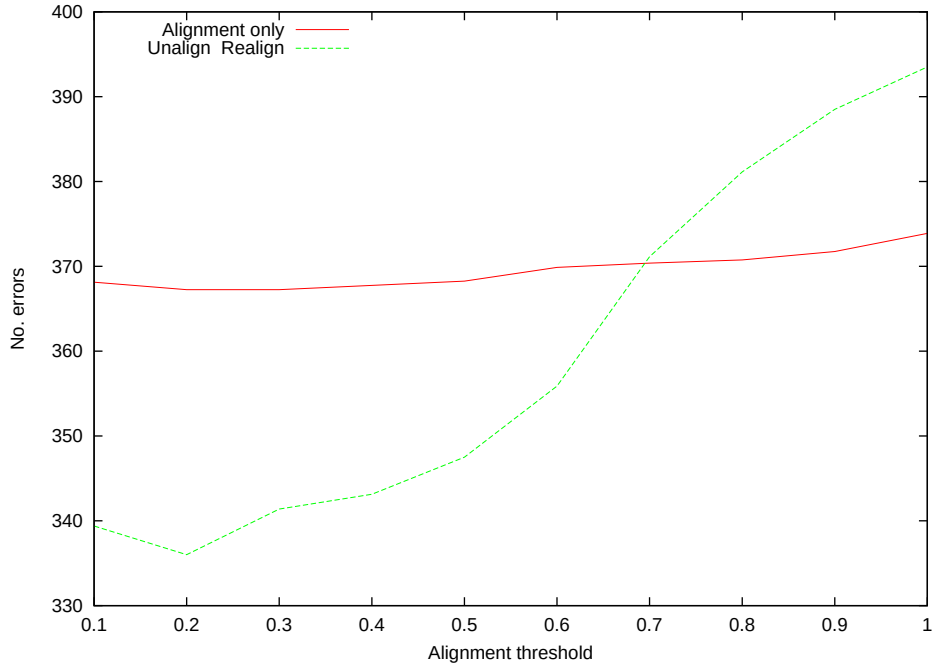


Figure 3.24: Performance of score-based alignment with and without unalignment stage at optimal alignment threshold.

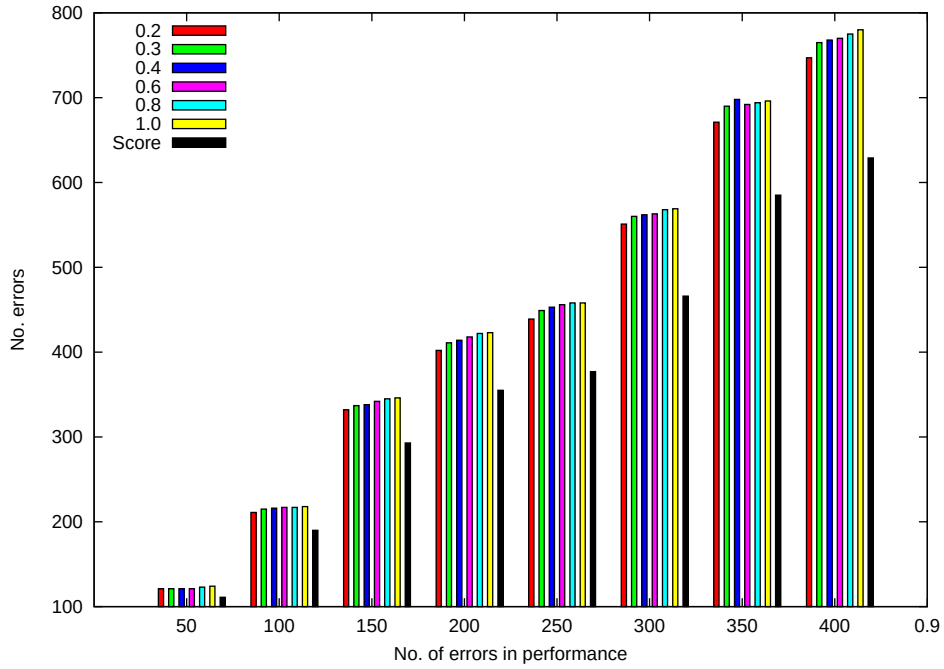


Figure 3.25: Score-based alignment vs Static alignment

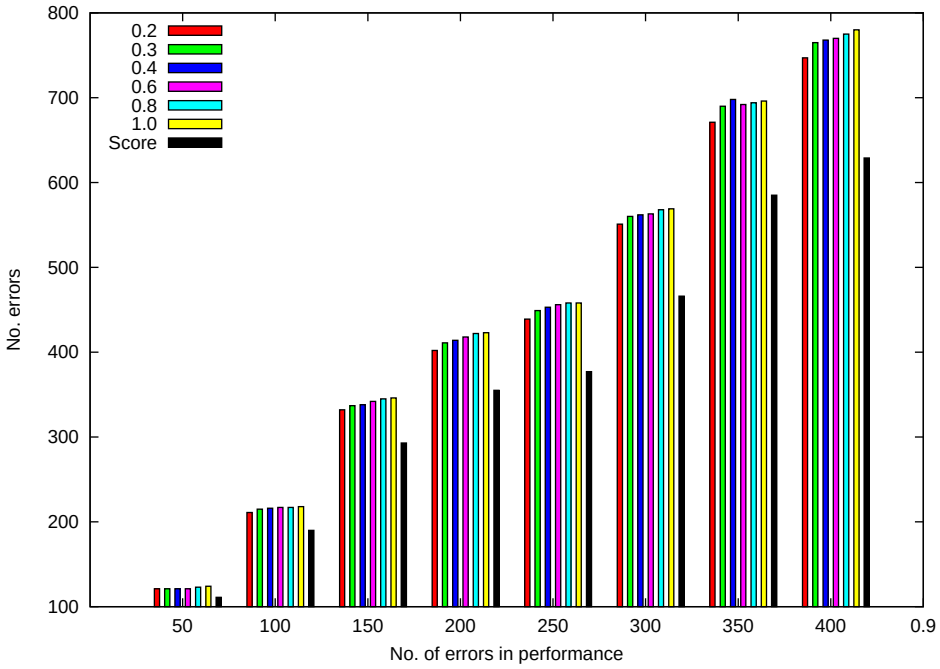


Figure 3.26: Score-based alignment vs Static alignment

3.7 Analysing real performance data

Results against artificial data have suggested that improvements suggested in this thesis can result in a considerable improvement in performance over existing matchers. In this section, different matchers are tested against real performance data. Section 3.7.1 will test the static and the dynamic matchers against the two real performance data. Section 3.7.2 will test the various interpolating matchers.

In the case of interpolating matchers, all performances have previously been matched using the dynamic matcher with a clustering threshold of 0.5. In Shostakovitch Prelude 5 two matchers will be tested: the static alignment matcher which uses the same method used in Hoshishiba [40] & the dynamic re-alignment matcher which removes false matches and creates new matches based on spline interpolation and score/performance analysis. These two matchers and one other will be tested against the excerpt from Chopin opus 66. The extra matcher which will be tested is the voice-independent realignment matcher. This matcher has one difference over the score-based realignment matcher. When re-aligning matches using spline interpolation, the voice independent matcher will use two spline interpolations, one for each voice/hand.

The error rates quoted in the graphs in this section quote an error which is the sum of two percentages. The false negative error is the number of correct matches which the matcher failed to identify as a percentage of the total correct matches in the authoritative version. The false positive error is the number of correct matches identified by the matcher which are not in the authoritative version as a percentage of the total number of correct matches identified by the matcher. The sum of these two percentages provides the overall error rate.

3.7.1 Static vs. Dynamic Matcher

In the analysis of Prelude 5, the static matcher was tested with different thresholds ranging from 0.1 to 0.4 secs (figure 3.27). The dynamic matcher was tested using thresholds between 0.3 and 0.9 (figure 3.28). The static matcher (Section 3.27) peaks at a threshold of 0.125 secs. At this threshold the matcher correctly identifies 97% of the correct matches with 2.09% of the matches made proving to be false, giving a combined error of 5.09%. The average tempo of the performance was 103bpm, which suggests a minimum ioi of just under 0.3 secs. Therefore there is no obvious relation between the tempo of the piece, the score content and the optimal static threshold.

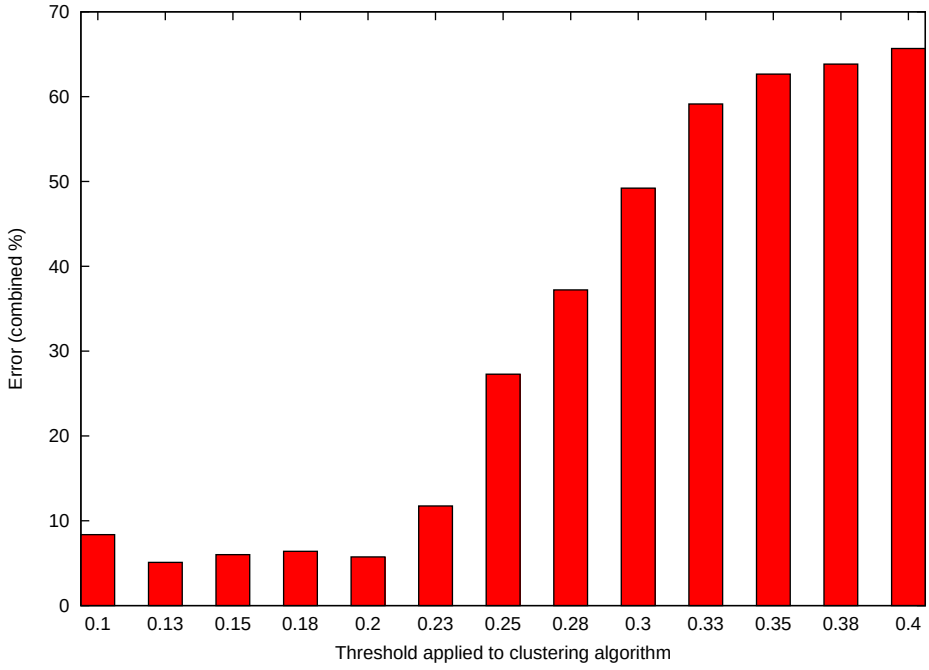


Figure 3.27: Performance of static matcher on Prelude 5

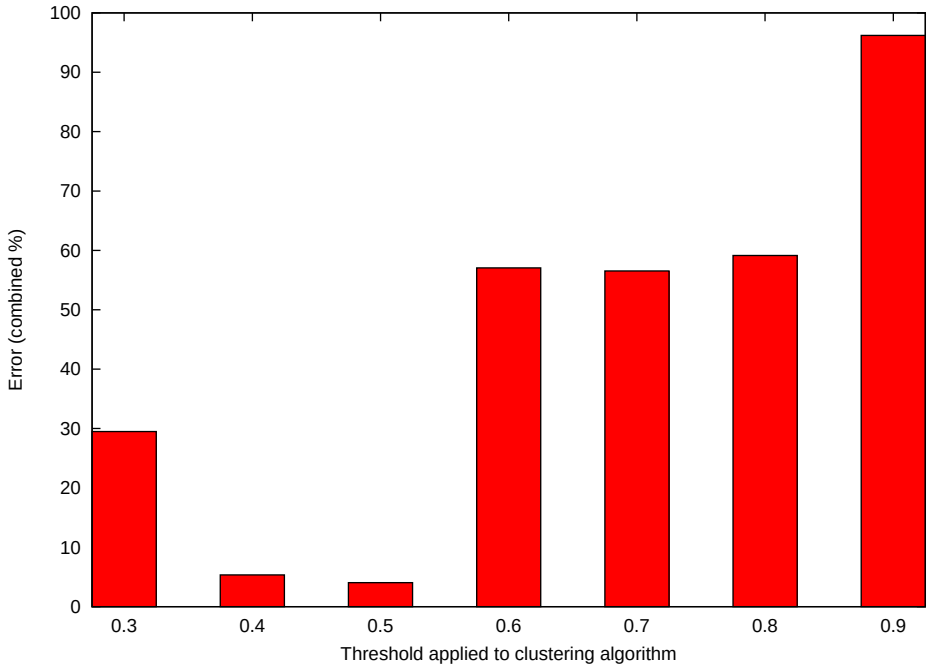


Figure 3.28: Performance of dynamic matcher on Prelude 5

The results of the dynamic matcher can be seen in Figure 3.28. As previously proposed, the dynamic matcher performs optimally using a strictness of 0.5. At this point it accurately identifies 97.52% of the known correct matches with only 2.5% false positive results, a combined error of 4.05%. As the threshold of the static matcher increases, the error rate increases steadily. Conversely, the performance of the dynamic matcher sharply declines when the clustering threshold is set too high. This is a result of the matcher taking a pathological route part-way through the analysis. It is possible this effect may have been minimised by increasing the size of the analysis window.

In the analysis of the excerpt from Chopin Opus 66, the dynamic matcher (Figure3.30) again outperforms the static matcher (Figure3.29). At its optimal settings, the dynamic matcherachieved a combined error of 30.8%. The average minimum ioi based on the average tempo of the entire performance is 37.5 milliseconds, although this value has less meaning in the context of this performance where it is known that there is considerable freedom between the timing of both hands. The static matcher actually performs optimally when the threshold was set to 0 which resulted in an error rate of 32.4%. At this threshold the static matcher performs no clustering of performance notes whatsoever. In

this situation, the static matcher is operating as a monophonic matcher, and it is only possible for the matcher to match a chord in the score to a single note in the performance.

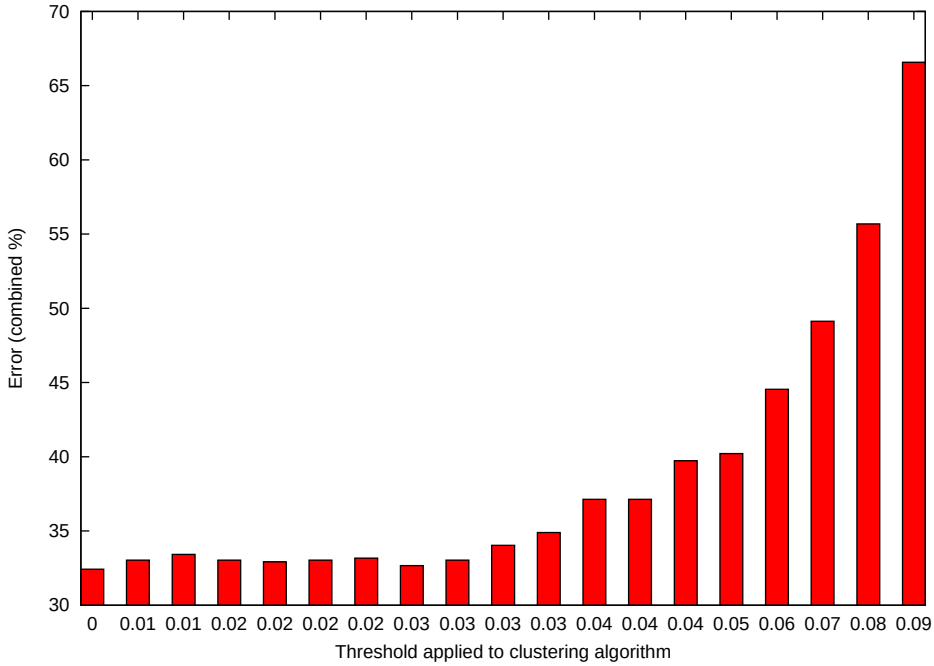


Figure 3.29: Performance of static matcher on Chopin excerpt.

In all pieces, the dynamic matcher proved to be more accurate than the static matcher. This is true with respect to both the number of correctly identified matches, and the reduction of false positives.

One of the hypotheses made before testing was that 0.5 would prove to be an optimal value of strictness for the dynamic matcher. While this is true in the analysis of Shostakovitch, in the analysis of Opus 66 it can be seen that the optimal value for the threshold of the dynamic matcher is not 0.5. However, the results gathered across the two real performances implies that 0.5 may provide the most consistent threshold, if not the most accurate in all cases. Consistency is an important consideration when the results of all matchers are considered. It is unlikely that the optimal thresholds for a piece can be identified before a piece has been analysed, and, as has been discussed already, the number of correct correspondences created does not reflect the number of *accurate* correspondences.

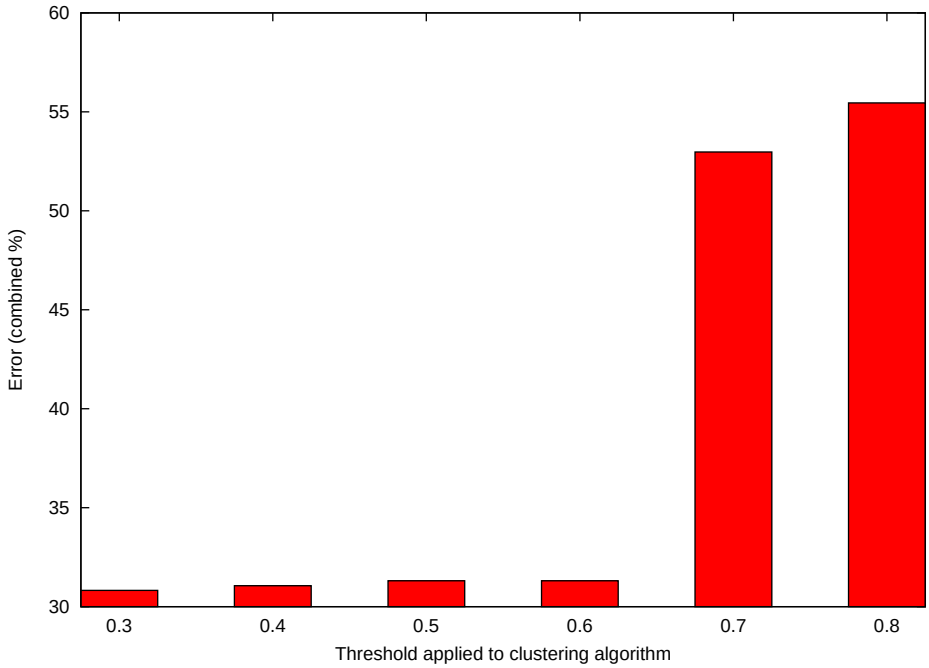


Figure 3.30: Performance of dynamic matcher on Chopin excerpt.

3.7.2 Interpolating matchers

This section tests various interpolating matchers against real performance data. All performances were first analysed using the dynamic matcher with a threshold of 0.5.

The static alignment matcher was tested at a range of thresholds between 0.125 and 0.9 (see figures 3.31 & 3.32). The score-based realignment matcher (figures 3.33 & 3.34) and the voice-independent matcher were tested using unalignment thresholds ranging from 0.2 to 0.9 and realignment thresholds from 0.6 to 1.0.

A realignment threshold of 1.0 indicates that the matcher will search for a matching note between the predicted times of the immediately preceding & following notes. A suitable match should not occur outside of the notes, however, at certain thresholds, the static alignment matcher will search for matches outside of this range. Therefore, as a final test the voice independent matcher was tested with realignment thresholds between 1.0 and 2.0 at two unalignment thresholds 0.2 & 0.5 (3.36).

Static alignment matcher

The dynamic matcher already achieved a very high match success when used to evaluate the Shoshtakovitch performance . In the tests performed, the static alignment matcher performs optimally at thresholds from 0.5 to 0.9 secs. These are the highest thresholds at which the test was run. Also, there is little variation in the results of the static matcher across different thresholds. The error rate ranges from 3.52 to 3.13. This suggests that there were very few matches available to be matched.

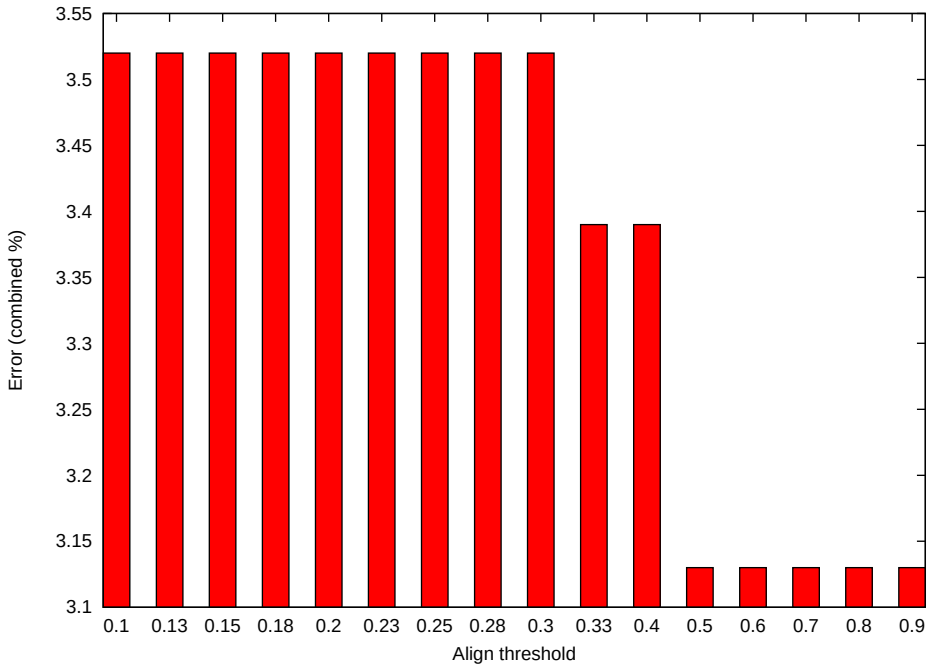


Figure 3.31: Results of the static alignment matcher evaluating Shostakovitch Prelude 5

When run against the Chopin excerpt, the static alignment matcher displays a far higher variation in results and a much higher improvement over the dynamic matcher alone. This is to be expected as the polyrhythmic nature of the piece will result in many out-of-order notes which DP matching cannot resolve. The optimal result is an error rate of 11.6 when the threshold is 0.325 secs.

Score-based realignment matcher

Figure 3.33 shows the results of score based interpolation on the performance of the fifth prelude. The performance of the previous interpolation algorithms were

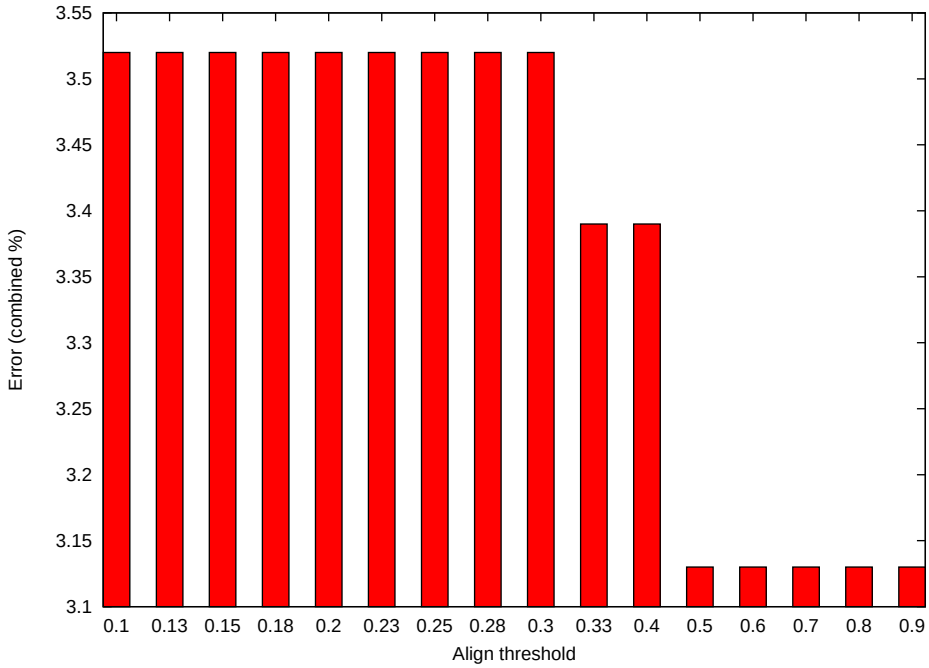


Figure 3.32: Results of the static alignment matcher evaluating Chopin excerpt

already very close to perfect and as a result, there is very little to improve upon. However, the score-based realignment matcher still improves on the already impressive match results. The optimal error rate is 0.78 which is achieved at unalignment/realignment thresholds of 0.3/0.9, 0.3/0.6, 0.2/0.8 & 0.3/0.8. As the unalignment threshold approaches 0.9, the error rate is in the same region as the rate achieved by the static alignment matcher. At this threshold, less notes are likely to be unaligned therefore the advantage of unalignment is quite clear from these results.

The dynamic realignment matcher does not perform so well when analysing the Chopin piece.The optimal thresholds are with unalignment/alignment set at 0.3/1.0. This achieves an error rate of 28.5. This is considerably worse than the static alignment matcher achieved at all thresholds. In fact, the matcher rarely performs better than even the dynamic matcher. Therefore the alignment algorithm is incapable of realigning matches which were originally matched by the dynamic matcher but removed by the unalignment process. For this reason, further tests were conducted using the voice independent matcher.

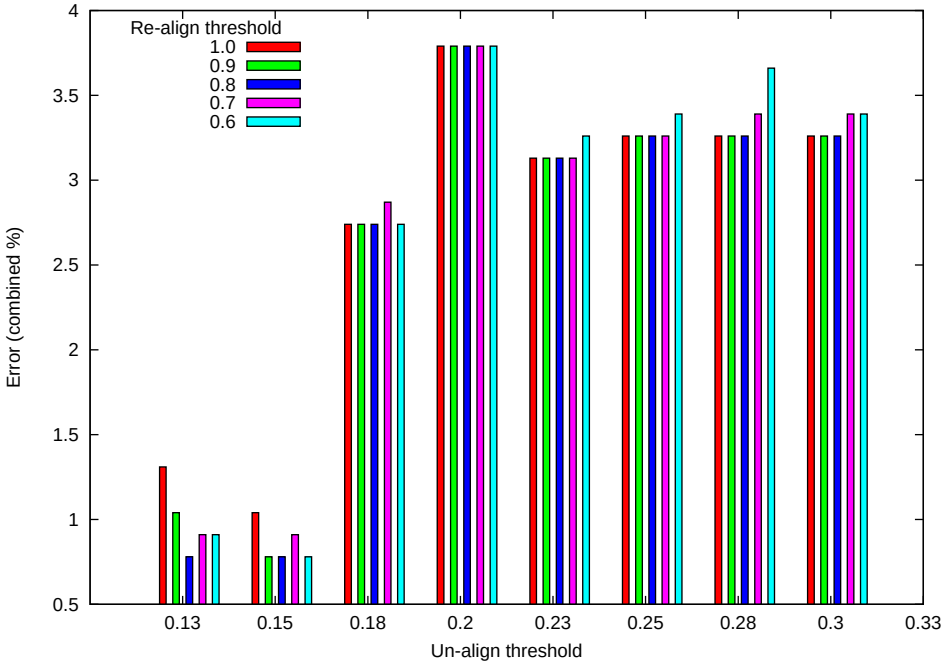


Figure 3.33: Results of the realignment matcher on Shostakovitch Prelude 5

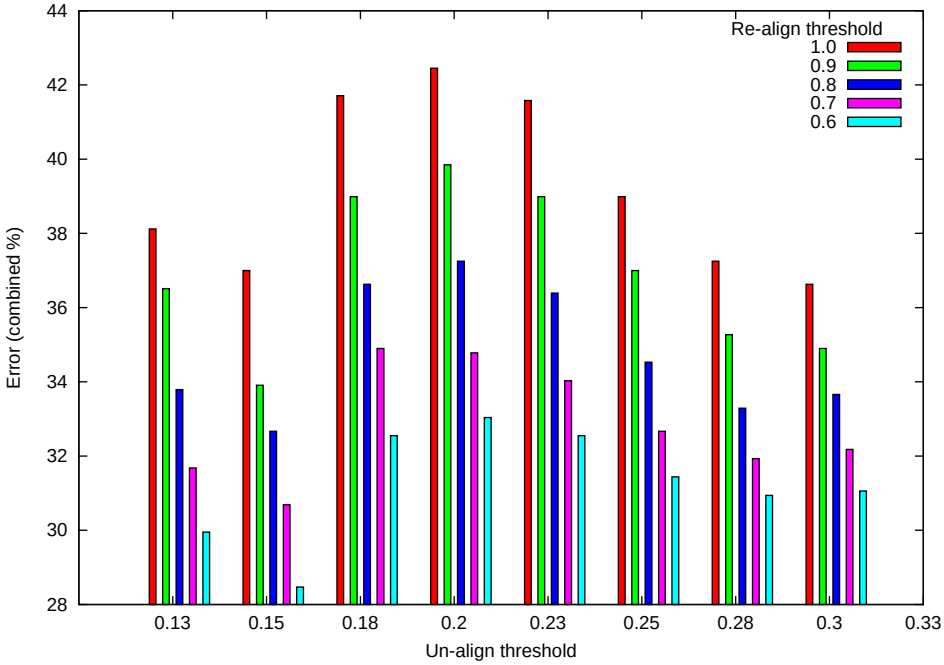


Figure 3.34: Results of the realignment matcher on Chopin excerpt

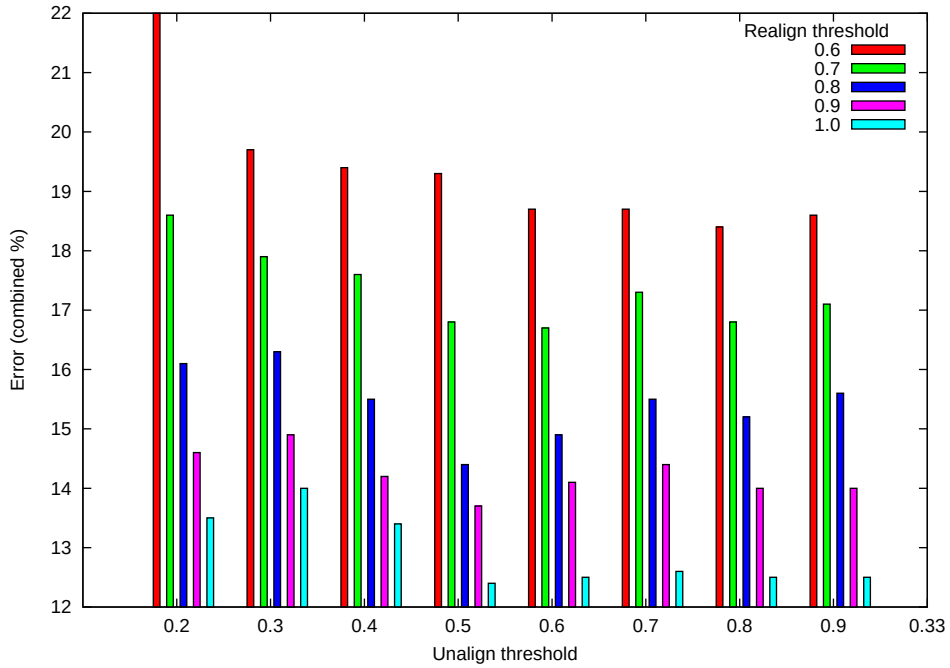


Figure 3.35: Initial results of the voice independent matcher on Chopin excerpt

Voice-independent matcher

The voice independent matcher performs unalignment and realignment on each voice/hand independently The first analysis of opus 66 can be seen in figure 3.35. These results show a considerable improvement over the standard realignment algorithm which did not analyse both hands independently. However, the performance is still less than that of the first interpolation algorithm. The optimum error rate of 12.5 is achieved with unalignment thresholds of 0.6 to 0.9 and with the highest alignment threshold 1.0.

These results suggest that increasing the alignment threshold over 1.0 might provide better results. This searches for notes outwith the time they should occur according to tempo estimates. However, the static alignment matcher will also search outwith these ranges. Therefore, figure 3.36 displays the results of analysis at a range of alignment thresholds from 1.1 to 2.0 and at unalignment thresholds 0.2 and 0.5.

When the alignment threshold is raised above 1.0, the voice independent matcher now performs similarly to the static alignment matcher. The optimum error rate occurs at unalignment threshold of 0.2 and alignment thresholds of 1.5 & 1.6. Th

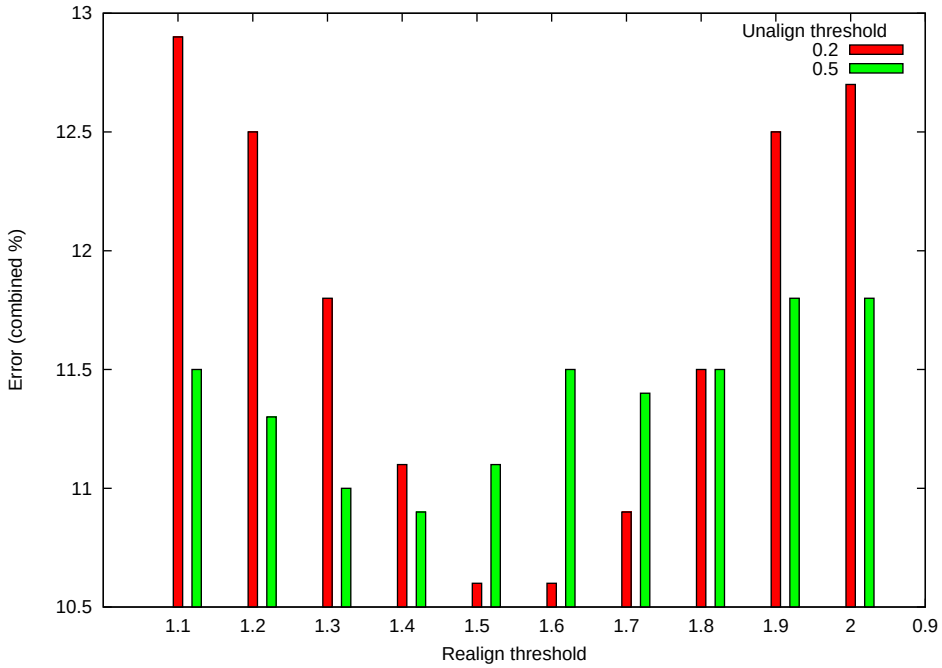


Figure 3.36: Further results of the voice independent matcher on Chopin excerpt

3.8 Conclusions and further work

This chapter has presented several new methods to improve the process of matching musical scores to performances of those scores using Dynamic Programming. The explanation of the application of Dynamic Programming to score-performance matching is more detailed than others in previous literature. There are many subtle differences between various DP matchers, and only with such a detailed examination can we be sure which method is most suitable to score-performance matching. This examination highlighted the inadequacies of the DP method. In light of these inadequacies, the following improvements to the process of score performance matching have been made:

1. Improved similarity & cost evaluation formulas for grid population (section 3.3.2).

2. An new algorithm for clustering performance notes prior to DP matching based on score & performance analysis (section 3.3.1).

3. An algorithm for removing false positive matches from a DP evaluation.

4. An improved algorithm for aligning new matches after DP evaluation.

5. A voice-independent unalignment/realigning matcher which improves performance on polyrhythmic pieces.

Previous matching systems have all relied on static, time-based thresholds to cluster notes into chords (Dannenberg [17], Hoshishiba[40]).The dynamic matcher is the first score-performance matcher known to the author to use score information to dynamically adjust the parameters of the matching process. This information is used in the process of clustering performance notes into chords, which significantly increases performance and resistance to pathological errors. Without this information, the clustering process is unable to adapt to fluctuations in tempo or changes in the density of notes within the score. The tests on artificially generated performances (section 3.6.1) and those conducted on real performances (section 3.7.1) demonstrated that dynamic clustering vastly improves the performance of the matching process.

The analyses of real performances (figures 3.28 & 3.30) tested a range of clustering thresholds. In the case of Shostakovitch, the error rate dropped by over 50% when using dynamic matching as opposed to static matching. In the Chopin excerpt, the error only improved by 7%. However, the Chopin excerpt was chosen because the polyrhythmic nature of the piece posed considerable difficulty to the DP matching process. The performance of the static matcher improved as the threshold approached 0 seconds, at which point the static matcher is not actually matching polyphonically at all.

In both cases, the optimal static threshold was not related to the average tempo of the piece. Therefore choosing an optimal threshold is extremely difficult. As discussed, the number of matches identified by an evaluation does reflect the number of accurate matches. However, this is the only method of choosing a clustering threshold for static matching.

These tests showed that the dynamic matcher is more robust over a range of thresholds. It was proposed that a threshold of 0.5 would prove to be a suitable threshold for most performances. In the small test set this has proven accurate. The spread of chords is controlled in artificial tests, therefore this can only be tested on real performance data and more tests will confirm whether this assertion is accurate.

It was shown in theory that there are certain shortcomings inherent in the DP matching method and it is incapable of resolving certain situations due to a lack of temporal information. Hoshishiba's system [40] is the only example of a previous matcher which attempts to resolve errors created during the DP matching process. This system used spline interpolation to attempt to align new matches in situations where performance notes occur out of sequence in

relation to the musical score. The static threshold used to select notes to align was unable to adapt to changes in tempo and note density during the course of a performance. Therefore a new method was developed which dynamically adjusted thresholds based on the instantaneous tempo of the piece and the properties of the music at that moment as extracted from the score. Tests using artificially generated performances demonstrated a consistant improvement in performance using thresholds based on tempo and score information.

The investigation into the inherent shortcomings also demonstrated that creating new correspondences is not capable of resolving all of the shortcomings of the DP method. To achieve this, false positive matches must be removed from the evaluation before searching for correspondences based on temporal information. An algorithm using the same techniques as the new algorithm for aligning new matches was developed. The effect of removing false positive matches before aligning new matches was tested. Tests on artificial performances showed that the removal of false positive matches improved the final evaluation across all tests. The consistency of the performance of the alignment only matcher across various alignment thresholds demonstrates that without removing false matches, the effectiveness of searching for new matches is limited.

The method used by Hoshishiba [40] is analogous to the static alignment matcher which creates new alignments using a static threshold. In tests on real performances this was tested against the dynamic realignment matcher which removes false matches and realigns new matches based on dynamic thresholds. In the tests on Shostakovitch Prelude 5, the static alignment matcher achieved an error rate of 3.13% whereas the dynamic realignment matcher achieved an error rate of 0.78%. The analysis of the Chopin excerpt proved less conclusive, and the dynamic alignment matcher proved only slightly more effective than the dynamic matcher alone, and significantly less effective than the static alignment matcher. However, the excerpt is a polyrhythmic piece and there is considerable difference in timing between the parts for both hands. Therefore a a small adjustment to the dynamic realignment matcher which analyses voices/hands separately resulted in an optimal error rate of 10.6%, a 10% decrease compared to Hoshishiba's method.

A summary of the optimal performance of all matchers for the Shostakovich and Chopin examples is shown in figures 3.37 & 3.38 respectively. These show that the improvements developed and presented in this thesis exhibit improvement over previous methods.

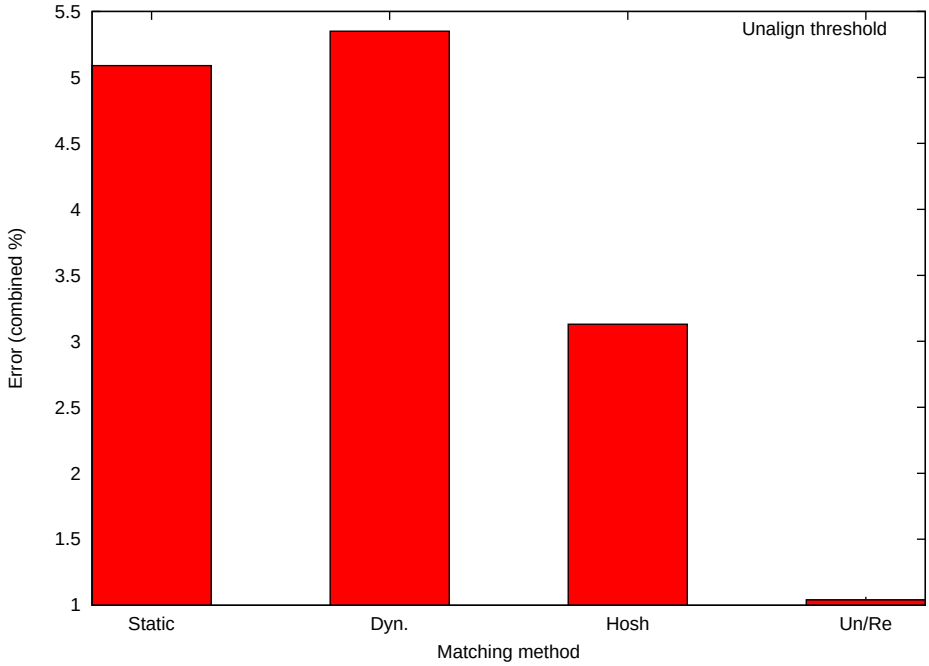


Figure 3.37: Optimal error rates for all matchers tested on Shostakovich Prelude 5

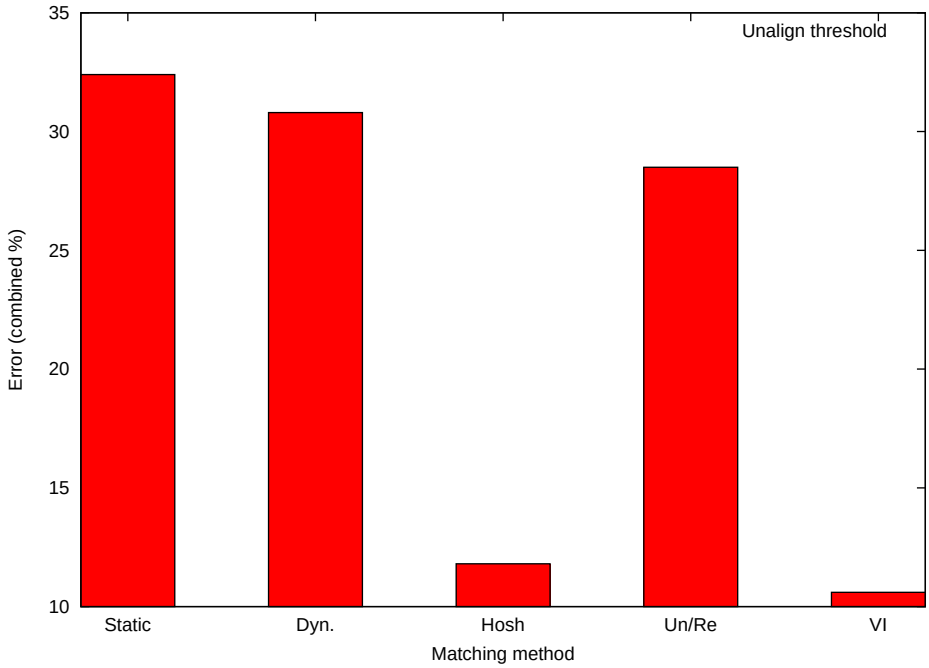


Figure 3.38: Optimal error rates for all matchers tested on Chopin excerpt

Further work

These methods would still benefit from yet further testing. Although the author is confident of the performance and reliability of the algorithms presented here, further tests may provide a better understanding of the parameters which are most likely to achieve a near optimal output based on properties of the piece such as note density, tempo & score directions.

The results have shown that the number of matches is not an accurate indication of the performance of a matcher. In a practical situation, where the results will be used to analyse performance issues, the presence of an incorrectly matched pair can cause the same disturbance in the results of further analyses as that caused by a missing correspondence.

The interpolation method currently calculate splines once. The performance may be improved by recalculating the spline after the inclusion/exclusion of each new/removed match. There are situations where expressive timing have caused a sequence of notes in one voice to be missed by the DP matching algorithm. The spline is calculated from the two correct correspondences. Therefore, when a new correspondence has been found, it could be used to create a more accurate spline for the analysis of surrounding unmatched notes.

To provide reliable data on which to base further musicological analyses, the correspondence between the score and the performance should be 100% accurate. To achieve this requires an improvement in the ease and speed with which correspondences can be evaluated and corrected manually. This can only be achieved with an adequate tool for visualising correspondences in the context of the score and the performance. Therefore a tool which displayed the musical score, the performance data (for example a piano-roll style display as found in many sequencer applications) and played audio or MIDI synchronously would greatly increase the ease with which correspondences can be manually assessed and adjusted. A visualisation such this would also benefit the analysis of performance issues in general.

The improvements in clustering presented in this thesis could be applied to score-performance following using DP. As it has been shown in the discussion on the direction of path evaluation (section 3.2.5) a forward path evaluation can reduce 'future match' errors. This requires a reverse grid population routine which, in the case of score-performance following requires the entire grid to be calculated each time new performance notes are added. The advancing window method used to dynamically adjust clustering thresholds removes the necessity to calculate a grid based on the entire score/performance.

Chapter 4

Analysis of microtonal performance

Microtonal music, where the scale is broken into more than twelve divisions, or those divisions are different from the normally accepted ones, is usually performed on bespoke instruments such as fretted or keyboard instruments, or synthesised using software or hardware electronic instruments. Other instruments which do not inherently quantise the pitch of the sound produced require the performer to use auditory feedback to maintain accurate pitch during performance. This necessitates a knowledge of the intervals and frequencies of the tuning systems which often combines elements of both sensory and muscle memory. In the performance of microtonal music, the frequencies which relate to each pitch and the intervals between these pitches is completely different to those of the 12 tone equal tempered scale. Therefore, performing microtonal music demands that the performer 'retunes' the ear to cope with vastly different tunings.

The first stage of the *Microtonalism* project [1] [31] aimed to investigate the feasibility of performing microtonal music on conventional instruments and through singing, and to create tools which might assist in the process of retraining and rehearsal.

The output of the project included, firstly, a small corpus of pieces written by Professor Graham Hair (Professor of Music, University of Glasgow; Adjunct Professor Australian National University, Canberra & Visiting Professor Radford University, Virginia) in a 19-tone equal-tempered tuning system (19ET). These were performed by Ingrid Pearson (Clarinetist, Deputy Director

[1] Microtonalism Home Page http://www.n-ism.org/Projects/microtonalism.php

of Research, Royal College of Music) and Amanda Morrison (Soprano, Scottish Voices, BBC Singers etc.), and Graham Hair himself. Secondly, a set of software tools which assisted in the training and rehearsal process. The first of these tools, the *Rosegarden Codicil*[2], provided real-time and offline pitch feedback within a popular MIDI sequencer application. It is described in Section 4.1. The second tool, which will be discussed in Section 4.2 is an offline performance analysis application which uses a novel method of vocal segmentation to produce a representation of a vocal performance using PML. This provides a more detailed analysis of the performance and visual feedback in the form of an annotated score. This tool can be further developed to provide profound analytical information about a vocal performance.

4.1 The Rosegarden Codicil

The *Rosegarden Codicil*[3] is a microtonal rehearsal and training aid integrated into the powerful and popular open-source MIDI sequencer Rosegarden. Although the dysfunctions of MIDI have been stressed many times in this thesis (and in many other sources) the use of an audio and MIDI sequencing application in this instance offered benefits which outweighed the cost of subverting some of the standard internal workings of the sequencer. Namely, Rosegarden offered an engine which performed audio and MIDI recording & playback, and a graphical notation engine which exceeds most, if not all, comparable sequencers.

However, as with all MIDI sequencers, Rosegarden's representation of pitch is heavily influenced by MIDI itself. One of the key design principles in Rosegarden is to represent information as closely as possible to the format required for playback. In an application designed for real-time operation, this is a worthy goal. Unfortunately, this leads to considerable problems when adapting Rosegarden to accommodate microtonal music. Rosegarden represents pitch as a combination of MIDI note and accidental. Although it is possible to transform between 'integer + accidental' and the note-octave system, and thus both contain the same information, the notation engine and sequencer engine are designed purely for traditional 12ET mapping of MIDI integers and pitches i.e. C3=60, A4=69. To add full support for microtonal notation and MIDI playback would require considerable development.

The modifications made to Rosegarden include support for further accidentals: semi- and sesqui- (one and a half sharps, or between sharp and double

[2]http://cmt.gla.ac.uk/website/projects/rosegarden_codicil.html

[3]The name is a mildly tongue-in-cheek adoption of the legal term for an addition or supplement to a will (from the Latin *codicillus*, 'little book').

sharp) sharps and flats. However, this support within the core of Rosegarden is limited to simple editing within the graphical notation editor (the effect of transpositions or attempted playback on notes which use these accidentals is stable, but incorrect). The notation editor was modified to include a real-time pitch tracking widget[4] which displays a graph of pitch error (Figure 4.1). Visual feedback has been shown to aid the learning process when people are developing pitch accuracy [111]. The widget analyses incoming audio independently of the main sequencer engine to obtain estimates of the fundamental frequency. The incoming audio is fed through a ringbuffer which discards the oldest audio information, ensuring that the frequency estimate uses the most recent frame possible. Unlike other audio applications such as recording, reducing the latency is far more important than processing every single audio sample.

These estimates are used to plot a graph of deviation from the target pitch below the score which is updated during the performance. This provides immediate feedback to the performer during training or rehearsal, allowing adjustments to be made during the performance. A screenshot of the real-time tracker in action can be seen in figure 4.2.

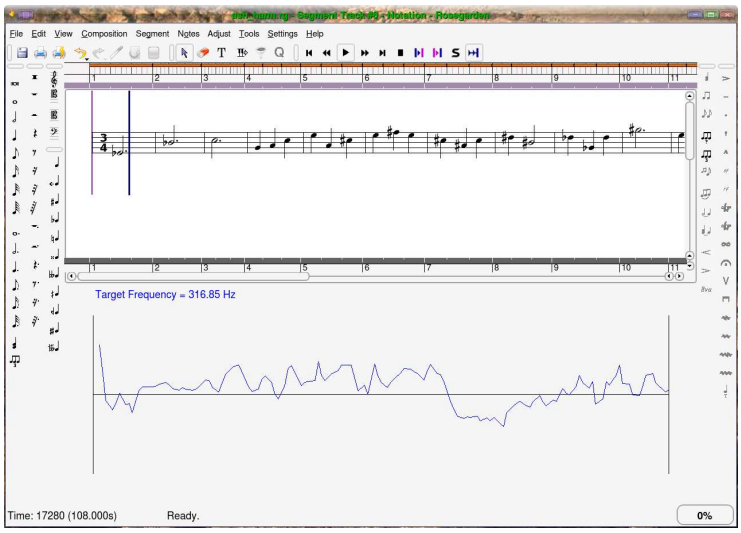


Figure 4.1: Real-time feedback of pitch accuracy in the Rosegarden Codicil

In addition to real-time feedback, a new notation editor window was developed which analysed a pre-recorded audio segment in the context of the score. Below the score, a series of graphs represented the deviations from the notated pitch. Rosegarden allows several audio files to be recorded in a session which

[4] A widget is a reusable component of a graphical user interface.

means that several rehearsals of the same piece can be recorded and compared. An example session for the post review is in Figure 4.2.

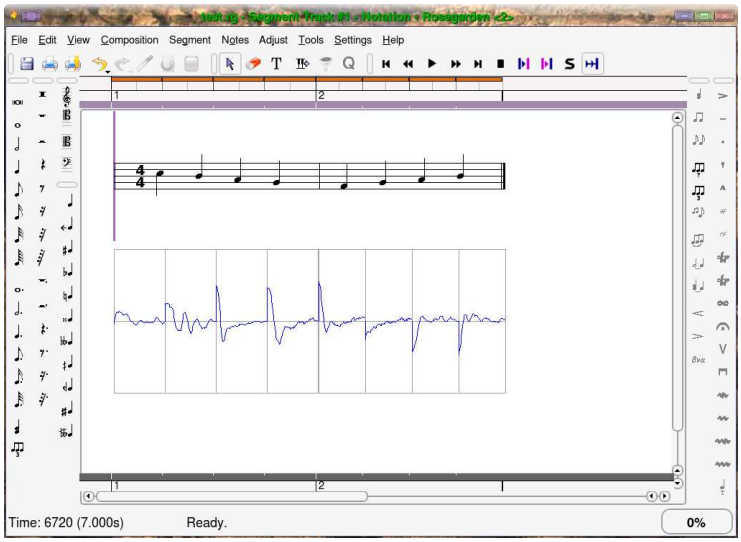


Figure 4.2: Offline review of pitch accuracy in the Rosegarden Codicil

The Codicil is capable of notating and tracking a wide variety of octave-based tunings[5]. The only known limitation of the system is the available accidentals, which effectively limits the maximum number of divisions of the octave to 63.

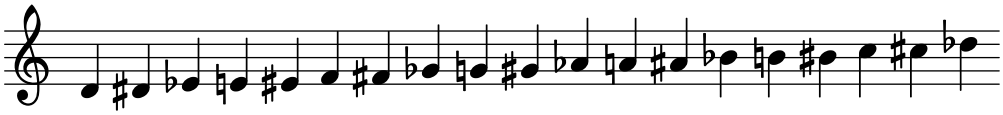
$$7 \text{ pitch classes} \times 9 \text{ accidentals} = 63 \text{ pitches maximum}$$

Results

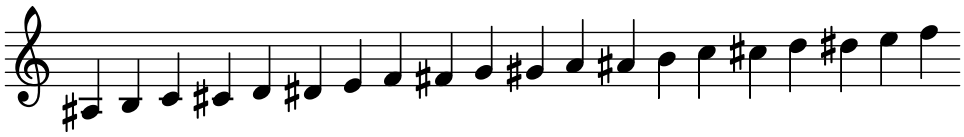
The Codicil was used extensively in the rehearsal of several pieces composed by Professor Graham Hair. These pieces were performed using combinations of soprano, clarinet and harmonium. The harmonium consisted of a traditional MIDI keyboard and a software synthesiser in which each MIDI note was re-tuned to correspond to the 19ET scale. This resulted in an octave which spanned 19 keys on the keyboard. To aid performance on a standard keyboard, the scores were translated into 'scordatura' or tablature versions of the score. The scordatura version contained the pitches which corresponded to the correct keys on the keyboard (as if playing in 12ET) which should be played rather than the sounding pitch. This means that on the keyboard, there is (at most) one note which represents the same pitch and key in both tuning systems. Usually this note was A4 440Hz, however due to the limited range of playable notes enforced

[5]The following release will support non-octave based tunings.

by the size of the MIDI keyboard this was not always possible. The A one octave above that would be 19 keys higher than A4 rather than 12, therefore in the scordatura version this pitch would be represented by F♯/G♭. The examples in Figure 4.3 demonstrate a short excerpt from a piece written in concert pitch, and scordatura.



(a) Concert pitch



(b) Scordatura pitch

Figure 4.3: A chromatic scale in 19-tone equal tempered scale shown both in actual notation (a) and scordatura notation (b).

4.2 Microtonal Performance Analysis

The Rosegarden Codicil provides real-time feedback of pitch accuracy, and the facility to record multiple performance to analyse the pitch accuracy offline. Although this tool was useful and accessible, there were several limitations to its usage. The process relied on synchronicity with the MIDI sequencer, therefore the performance had to be in strict metronomic time. Although the sequencer allows tempo to be adjusted, a performance will never exactly match the attack and release of notes as they occur in MIDI. A vocal performance cannot change pitch instantly, and a human performer is unlikely to have the accuracy to make transitions at exactly the time specified by the MIDI sequencer. Therefore, artefacts are visible at note transitions. These transitions are expected, and do not interfere with human visual analysis because we can isolate the steady section of the note. However, they would affect an analytical assessment of pitch accuracy. Therefore, it was necessary to develop a tool which provided a more detailed offline analysis of pitch accuracy, which did not rely on strict metronomic time. This would allow the analysis of a performance under 'normal' performance conditions i.e. it would have the ability to cope with expressive timing.

For this reason, tools were developed to assist in the analysis of microtonal

performance using PML. This consisted of a transcription algorithm specifically designed for microtonal vocal rehearsal and performance, and tools to assist in the visualisation of pitch accuracy.

4.2.1 Characteristics of performance in microtonal rehearsal

In the rehearsal situation, singers will often sign melodies without voicing the lyrics. Singing a constant vowel (typically 'ah') as opposed to singing lyrics prolongs the pitched section of the note which increases the amount of auditory feedback. Maintaining a constant vocal chamber reduces technical difficulty and a constant vowel produces a clearer tone increasing the quality of auditory feedback. The characteristics of a vocal performance in a microtonal rehearsal situation differ greatly even from the characteristics of rehearsal of standard repertoire. Analysis of vocal rehearsal as part of the Microtonalism project has shown that the average pitch of notes can be very inaccurate. This is to be expected, even in highly trained professionals who have perfect pitch. Also it is common for the frequency trajectory of individual notes to be very unstable. This is due to adjustments the performer makes to correct the pitch in real-time. These adjustments are usually as a result of either an analysis of the harmony between their performance and the accompaniment. The use of visual feedback regarding pitch accuracy exacerbates this problem.

4.2.2 Vocal Segmentation algorithm

There are a plethora of transcription or algorithms available in academia and commercially. Typically these transcription algorithms can be described as three processes: feature analysis, segmentation & quantisation [4, 53]. The algorithm presented here is a *segmentation* algorithm rather than a transcription algorithm. In other words, the algorithm is designed to segment a performance according to certain criteria. This algorithm is intended to be used to identify the regions of stationary pitch within performed notes, for the purposes of analysing pitch accuracy. Although this is similar in many respects to the first two stages of automatic segmentation, there are very important differences. This application does not require an exact onset which corresponds to the beginning of the note (whether that refers to the perceptual onset or the physical onset). Notes consist of unpitched sections, often within the attack of the note which should be considered when identifying the onset of a note. However, these sections are disregarded when identifying notes solely on the basis of frequency.

Additionally, within the context of microtonal rehearsal, many of the reliable

cues used for automatic transcription, or other types of segmentation algorithm are missing. The two most important cues used in transcription are pitch and dynamics. In vocal performance, it is very common for this dynamic information to be absent from note transitions. Performances are often sung legato where the transition between notes is only indicated by a change in pitch rather than dynamics. Therefore, an application which aims to transcribe vocal performance within a rehearsal environment must be capable of detecting transitions between notes solely on the basis of pitch information.

Unfortunately, in the context of microtonal rehearsal, pitch is also not as reliable as it is in the transcription of standard repertoire. As explained in Section 4.2.1, the pitch of a note is likely to be very inaccurate, and possibly very unstable. There are two common methods of identifying note transitions using only pitch information. The first, rather simple method, identifies transitions between notes at the boundaries between different pitches. This method effectively quantises the trajectory of frequency values into a sequence of pitch values, or scale intervals. This is often used in simple monophonic transcription routines such as those used to convert audio files into MIDI data.

The algorithm of Weihs and Ligges [110] uses such a method. The frequency trajectory is initially quantised to obtain a sequence of pitches. This 'pitch' trajectory is then smoothed to suppress vibrato. The resulting list of pitch values is then segmented and quantised in time to produce a transcription. This method is only of use in the segmentation of accurate performances. For example, if the frequency trajectory of a note crosses a pitch boundary, the note will be segmented into two separate notes rather than one (extremely out-of-tune) note. Furthermore, it discards the actual frequency information which is necessary in the analysis of pitch accuracy, or intonation.

The second cue for detecting note transitions based on pitch is the pitch gradient. However, vocal performance often contains vibrato which results in continuous oscillation in the derivative of the frequency trajectory. This vibrato must be suppressed to detect transitions between notes. Rossignol et al [86] describe several methods for the detection and quantification of vibrato in musical performance based on spectral modelling, spectral envelope distortion, auto regressive prediction and a method based on the analysis of maxima and minima in the vibrato. Those based on the spectrum of the signal, rather than the frequency trajectory will be affected by the presence of other instruments in the recording, whereas those based on the frequency trajectory (the latter three methods) can be applied to the output of any transcription algorithm. Of all the methods, the most promising method appears to be the maxima-

minima based method, though it has yet to be tested in the current application. This method first detects the maxima and minima of the frequency trajectory. Interpolation is then used to determine the two trajectories which follow the maxima and minima points independently. This method allows properties of the vibrato to be analysed such as frequency and amplitude. The geometric mean of the two calculated trajectories also predicts a frequency trajectory for the performance with the vibrato suppressed. This was used in a system for feature extraction and acoustic segmentation [85]. This system combined a speech/singing/noise discriminator optionally followed by the aforementioned vibrato suppression (for musical segmentation) and a multi agent segmentation algorithm. The segmentation algorithm comprised algorithms analysing 9 features including derivatives of frequency trajectory and energy, inharmonicity, a voicing coefficient, probabilistic pitch transition and AR modelling.

McNab et al [53] presented an automatic transcription system developed and used in applications requiring vocal transcription. The system describes an algorithm which segments vocal frequency trajectories. The system advances a window of analysis through the frequency trajectory. When a segment has been found of length 100ms in which all frequency estimates are within 50 cents of the mean frequency of the window, the segment is assumed to correspond to a note. The boundaries of the segment may then be extended as long as the previous criteria holds true. In many ways, this algorithm could be easily adopted to microtonal use. The threshold of $\pm$ 50 cents relates to the smallest interval in 12ET. This could be adjusted for each microtonal tuning. Also, the threshold is based on the mean frequency of the segment, which means that the algorithm is to some degree less affected by notes which are performed out of tune. In fact, the paper mentions the application of the system to transcribing just and Pythagorean tuning, though there is no mention of any changes to the thresholds for that purpose. However, the algorithm seems to have little accuracy in identifying steady portions of a note. Indeed, the algorithm identifies a glissandi as a series of segments which is contrary to the requirements of both microtonal rehearsal and automatic transcription.

4.2.3 Method

The piecewise linear segmentation algorithm can be subdivided into three stages:

1. Fundamental Frequency Estimation
2. Preliminary Onset & Endpoint Detection
3. Onset & Offset Localisation

Fundamental Frequency Estimation

The pitch detection stage analyses the audio stream producing a list of frequency estimates. The pitch detection algorithm is a two stage algorithm based on the autocorrelation method which, during the use of the Rosegarden Codicil, was found to give more stable results for the analysis of soprano voice over other simple methods such as the Harmonic Product Spectrum (HPS) [90]. The autocorrelation is calculated using the Weiner Kinchen theorem (below), which for real-valued functions allows the auto-correlation to be calculated using the Fast Fourier Transform (FFT). It states that the autocorrelation of the time domain signal is equal to the Fourier Transform of the power spectrum $S(f)$ of the signal. The initial estimate of fundamental frequency is estimated at the maximum value in the autocorrelation function after the initial peak at $\tau = 0$.

$$R(\tau) = \int_{\infty}^{-\infty} S(f) e^{j2\pi\tau}\, df$$

The second stage performs a localisation of the fundamental frequency by analysing the phase difference between successive FFT frames. The phase difference between successive frames allows the frequency of the partial within that bin to be localised within the bandwidth of the FFT bin as long as it is the only partial present in that bin [23]. The calculation of the autocorrelation in the first stage means that the FFT data is already available from previous calculations. When there is no distinguishable peak, the frame is labelled as un-pitched.

Initial onset & endpoint detection

The second stage searches the pitch trajectory to determine initial estimates of note onsets and offsets. The frequency trajectory is first converted into cents to remove non-linearity. The algorithm then proceeds to search for onsets & offsets based on gradient. To avoid the false detection of offset and onsets, any vibrato must be suppressed. The algorithm iterates through the pitch trajectory calculating for each analysis window, an average gradient. The average gradient was calculated using a method of linear regression. This provides a 'line-of-best-fit' (LOBF) which represents an average linear trajectory which minimises the error using the least squares method between a point and the closest point to it on the line for all points in the analysis window.

The frequency trajectory was analysed with a window step of 256 samples and a sampling rate of 44100kHz. The onsets and endpoints were detected based on a threshold gradient of 77.5 cents per second analysing the frequency

trajectory with a line which spanned 100 frequency estimates. These values were also chosen through qualitative assessment, using performances of the 19ET pieces composed by Professor Graham Hair. A purely scientific examination of how pitch gradient relates to how humans segment music would require an in-depth psychological study. This should also include an investigation of the effect microtonal tunings have on pitch segmentation. This is beyond the scope of this project, therefore a qualitative assessment must suffice until such work has beecarried out.

When the gradient of the LOBF fell below the threshold, an onset was recorded at the *beginning* of the analysis frame. Conversely, when the gradient rose above the threshold, an offset was recorded at the *end* of that frame. This ensured that the recorded onset was consistently judged to be before the apparent onset of a note, and the recorded offset was consistently judged to be later than that of the actual offset.

Localisation of note boundaries

To localise the onset and endpoint points of each note, note candidates are created based on the initial onset and endpoints discovered in the previous stage. The localisation process also relies on linear regression and each note candidate is represented by a LOBF which is calculated based on each point in the pitch trajectory which lies within the onset and endpoint points. Localisation was an iterative procedure which followed the steps below.

1. The error for a candidate note is calculated between a frequency estimate and the closest point on the candidate line to that estimate. The error is calculated using the least squares method.

2. The LOBF and subsequent error is calculated for the case where the last point is removed from the original set.

3. The LOBF and subsequent error is calculated for the case where the first point is removed from the original set.

4. If the removal of neither of these two points result in a decrease in the error of the line, the process is halted.

5. Otherwise, the point which caused the greatest decrease in error is removed.

6. This process is repeated until the potential decrease in error, as a fraction of the current error, is less than an arbitrary limit.

It is essential in the above procedure that the boundaries of the initial candidate are 'overestimated'. Due to the periodic nature of vibrato, there will be numerous sets of boundaries at which the error will exhibit a local minima. Therefore, the localisation must be performed using regression rather than extrapolation.

Appendix B shows the results of localisation. The diagrams show the analysis of the second phrase of *Ash* by Graham Hair which extends from bar 4 to the final A in bar 5 (see Figure 4.4). The initial candidate for each note is represented by a blue line and the final candidate after regression is indicated by the green line. Each unit on the y axis represents an interval of 63.158 cents (one chromatic step in 19ET) where 0 represents A=440Hz.

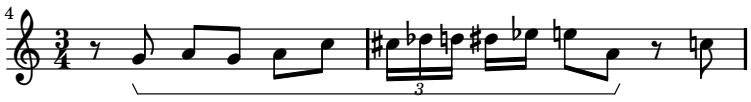


Figure 4.4: Bars 4 & 5 of *Ash* composed by Graham Hair. The second phrase of the piece is indicated.

In some instances, two notes were incorrectly analysed as one continuous note. This occurred only in instances where the notes are just one hyperchromatic step apart, and the transition between the note is unclear. In these instances, the trajectory of the individual notes tends towards the contour of the phrase at that point. For example, the example in figure 4.6 shows two notes which have been incorrectly identified as a single note. The trajectory of the notes is steadily rising which obscures the transition by reducing the size of the transition. These two notes occur in bar 6 (Figure 4.5) within a run of five notes each one hyperchromatic step above the other. Repeated rising or falling hyperchromatic steps often proved to be the most difficult passages for the performers. Also, in tunings with more divisions of the octave, the transition between notes will obviously be less clear than those with fewer divisions of the octave. Therefore, the threshold could be adjusted according to the tuning to achieve optimal performance. Tunings with larger intervals could afford a more liberal threshold, whereas tunings with smaller intervals will require a tighter threshold.

To reduce the number of incorrectly merged notes, notes whose trajectory spanned more than a threshold were split into two separate notes. The notes were split by finding the point within the original note's duration where the magnitude of the gradient was highest. At this point the line was split into two

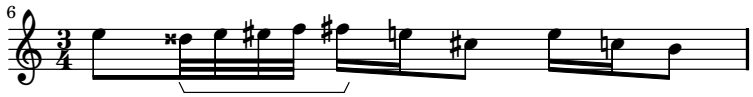


Figure 4.5: Bar 6 of *Ash* composed by Graham Hair. The bracket indicates the consecutive rising chromatic steps.

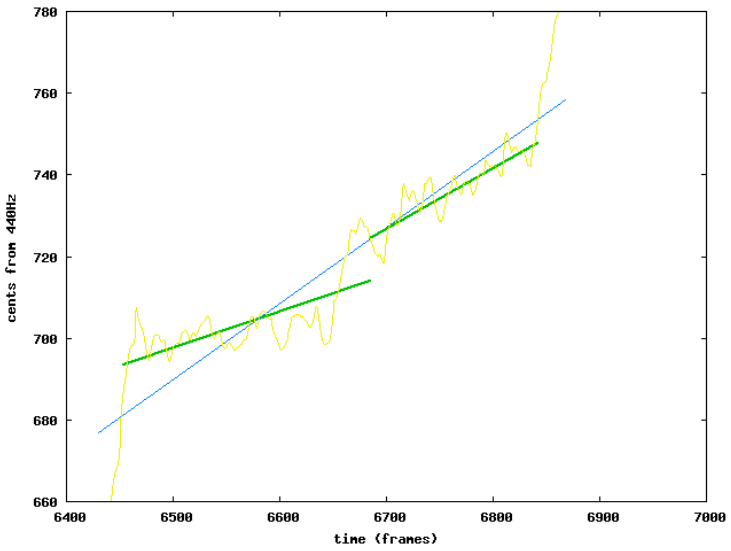


Figure 4.6: An example of the note splitting process. The blue line represents the original note candidate. The two green lines represent the two new candidates following the splitting process.

lines, and each line was regressed independently.

The ranges of some individual notes was found to be greater than that of some of the merged notes. Therefore, choosing the optimal threshold was a compromise between correcting the most errors and breaking the fewest correct identifications. In the course of this project, this threshold was found to be 0.8 of a 19ET semitone. However, the optimal threshold will be determined by the tuning system. In tuning systems with larger intervals, this process may even be ignored completely, as the note transitions should be clearer.

4.2.4 Results

To assess the performance of the segmentation algorithm, the output was tested against manual segmentations of the performance. Three listeners were asked to manually segment a performance using the Audacity audio editor [3]. The indi-

viduals were to identify segments which exhibited a steady pitch. The onset and endpoint of each segment were determined by repeatedly auditioning a segment and adjusting the start and endpoint. The listeners performed segmentation without the use of the score. All of the test subjects were competent musicians. However, a comparison of the results shows that even human listeners have difficulty in segmenting a performance in a microtonal rehearsal situation.

There were two situations where the human segmentations disagreed. In the first situation, a fluctuation in pitch during a transition (which did not correspond to notes in the score) was judged by one listener to contain two very short notes, whereas the other two listeners judged the fluctuation to be too unstable to correspond to notes. The algorithm identified one segment within this fluctuation.

At the second point of dispute, two listeners judged one note to be two distinct notes. The remaining listener and the algorithm both identified one segment at this point. Besides these cases, the algorithm identified three extra segments, and failed to identify one segment. This section corresponded to one single note within the score. There is a discernable change in pitch, although whether this this change is sufficient to constitute two separate notes is obviously contentious. To allow comparison across all subjects, the segments which corresponded to disputed sections were removed from the calculations in every segmentation list.

The mean absolute deviation across all common onsets and endpoints identified by the human listeners was 0.016 seconds and the maximum deviation found was 0.173 seconds. The mean absolute deviation of the algorithm's estimated times from the average human estimated time was 0.076 seconds, with a maximum at 1.2 seconds. Excluding this abnormality, caused by an extra segment which was removed from calculations, the maximum deviation is 0.720 seconds, and the mean absolute deviation is 0.072 seconds.

This is not considered to be a conclusive evaluation of the performance of such an algorithm. The listeners reported difficulty in discriminating segments based solely on pitch. They found difficulty in distinguishing the exact location of deviations in pitch in the presence of the deviations of other audible qualities such as timbre and dynamics. The graphical representation of audio information at the resolutions required for such a task also includes visual cues which may accompany a change in dynamics, pitch or timbre. The process of isolating an audio segment may also have psychoacoustic effects caused by removing the surrounding context.

However, where human segmentations agreed on the macro scale, the devi-

ation across listeners was actually quite low. The difference between *how* the performance was segmented i.e. which notes occurred rather than their exact times, demonstrates the difficulty of the task. The definition of a 'correct' segmentation in this context is difficult to determine.

A conclusive study would involve a psychoacoustic investigation of exactly where the perceived onsets and endpoint occur in passages which exhibit both vibrato and legato, and would certainly contain considerably more performances and listeners. Such a detailed analysis is outwith the scope of this study.

However, the aim of this project was not to perform accurate transcription. Rather, the aim of this project was to create a tool which would provide accurate analysis of performed pitch. An analysis of only the onsets showed that the algorithm's estimated onsets exhibited a mean deviation of 0.02 seconds from the average human predicted onset. The same comparison regarding the endpoints show a mean deviation of -0.06 seconds. Thus, the algorithm has a tendency to shorten the boundaries of note candidates at both ends. This conservative estimation of the boundaries of a note is desirable for segmentation for the purpose of analysing pitch accuracy because the aim is to find the steady section of a note rather than the perceptual onset and endpoints.

The results of the performance analysis were recorded in PML, along with the score. The matching algorithm of Chapter 3 was used to link the notes in the performance to their corresponding score notes. Using this information, an annotated score can be generated which provides a simple visualisation of pitch accuracy. Figure 4.7 shows a score annotated with pitch error. Where a performance note correctly corresponds to a note in the score, the performance accuracy is denoted above the score. The bars represent pitch accuracy plotted in the range +/- 31.579 cents (one 19^{th} of an octave).

Notes which do not have a corresponding note in the performance are displayed with red noteheads. As can be seen in the score, there are two sections where there are a significant number of wrong notes: bars & 9 and the final three bars in the performance. These sections are caused by pitch drift, where the performer has gradually lost her tonal reference. This has been identified as one of the main problems of microtonal rehearsal, as both the intervals themselves have changed, but also, in the case of 19ET at least, only A remains at the same frequency. All other intervals other than the octave are different from the 12ET system. Therefore the performer has few familiar references.

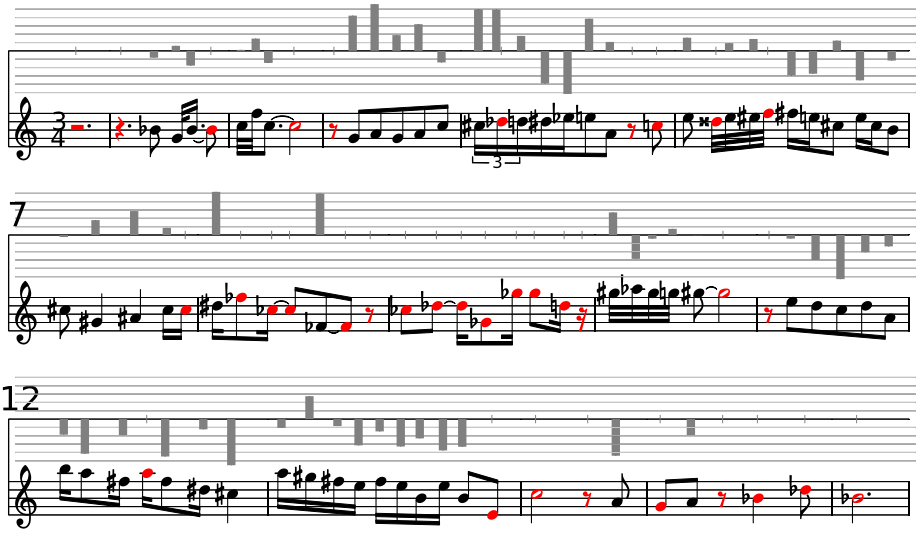


Figure 4.7: Annotated score displaying the deviation from exact frequency in a performance of the soprano part of Ash (composer Prof. Graham Hair).

4.2.5 Conclusions

Two tools have been successfully developed in the course of this project. The first is an easy-to-use microtonal rehearsal aid which provides real-time and post-performance feedback regarding pitch accuracy within an existing software sequencer application. The second tool provides feedback regarding pitch accuracy without requiring strict metronomical time which uses the technologies described in Chapters 2 & 3. The second tool has proven to be accurate with the small test set with which it was tested, however it would benefit from further testing. The segmentation algorithm has not been directly tested against other segmentation algorithms. The segmentation algorithm presented by Rossignol et al [85] has no test data. The paper which succeeds this paper [83] has results, but the test data is not specified, so a direct comparison is not possible. In any case, the aims of the two segmentation algorithms are different, Rossignol et al were attempting to segment features such as phonemes in addition to changes in pitch, and the pitch trajectories in that paper do not seem representative of the microtonal rehearsal situation because the transitions are clear and notes seem to have a very stable pitch trajectory. Similarly, the transcription algorithm presented by McNab [53] contains no results to which the current algorithm can be compared. However, the results of the algorithm here are consistent with human segmentation.

4.2.6 Further work

The algorithm presented here successfully segments a vocal performance for analysis of pitch accuracy in 19 tone equal temperament. The algorithm is not restricted to this tuning, however further testing may provide a deeper understanding of the parameters of the algorithms and improve performance in other tunings. The gradient threshold should be adjusted depending on the smallest chromatic interval present in the scale. For example, using 12 tone equal tempered music, the algorithm could be expected to perform even better because the higher chromatic interval causes a more pronounced pitch transition.

To improve usability, the algorithm would benefit from integration into a system which would provide the user with complete control over the analysis including the ability to select portions of the music and audition the performance at those points. This is possible immediately using the information within the PML file, and is awaiting the development of an adequate API for the representation for musical scores.

A system for the formal analysis of performance would require a rigorous study of the perceptual aspects, to ascertain exactly where onset and endpoints are perceived in complex situations involving gradual transitions involving pitch, vibrato and dynamics. Although this method is resistant to the presence of vibrato, the use of the vibrato 'removal' algorithm presented by Rossignol et al [86] may further improve performance and provide further analytical performance information.

Chapter 5

Conclusion

The aim of this work was to provide the framework with which issues relating to musical performance including the performance process could be investigated. The minimum amount of information required for this task consists of:

- The musical score
- Performance Markup
- Event-level correspondences between the score and the performance

A computer representation was presented which provided all the prerequisites above. It also provided the framework to allow the representation to be extended to include new structures within their own, natural hierarchy by providing relational links between internal structures, and the ability to reference locations in external files. This allows music to be represented in all it's manifestations simultaneously and synchronously. The use of existing file formats such as MusicXML, PCM audio files & GMS gestural files pulls together existing standards (without excluding new representations) which promotes the sharing of information and tools within the community. Together with the development of an API, this makes the representation immediately usable with existing tools. The representation is currently under consideration in the effort to create a standard for the representation of gestural information relating to musical performance [46, 47] which will include low-level streaming formats in addition to mid-level representations and high-level descriptions.

A new algorithm has been developed to analyse the event-level correspondences between a performance and a score. This algorithm contains a new approach to DP-based polyphonic matching and new algorithms for post and which builds upon the algorithms which preceded it. The new algorithm not

only showed an increased accuracy, it also showed an increase in reliability, especially in the presence of performance error and extreme expressive timing. This new algorithm complements the representation by allowing the third requirement above to be calculated from the first and second requirement.

The Microtonalism project described in Chapter 4 demonstrates a complete system in which PML is already being used.[1] This project itself presented a new algorithm for the automatic segmentation of vocal recordings. This algorithm, although aimed specifically for the transcription of vocal recordings in the context of microtonal rehearsal could prove a useful part in a system which is aimed towards accurate transcription if combined with analysis of the psychological aspects of segmentation of continuous pitch trajectories and the analysis of unpitched sound. The use of the representation and tools in this project demonstrates that the original aims have been accomplished. However, to ensure continuing development, it is necessary to foster a community which will allow the representation to evolve to meet the needs of future projects. The funded term of the project was successfully completed in February 2007, however, the project is ongoing in several institutions.

The success of the technologies presented here, and their application in current projects has highlighted the necessity for development in the visual presentation of musical information. Although a representation has been developed which allows various domains of music to be represented and analysed by computer, the application of this technology is still limited by the interface between the user and the information. The user interface, like the computer representation, must be capable of representing audio, performance information, analyses synchronously with the score. This is essential, not only to speed up the process of creating analyses, or interpreting the results, but also to allow access to musicians who do not have a background in computing. This requires an open API which will render a musical score on a computer screen. The example shown in Figure 2.22 is a small demonstration of how a small amount of information can be quickly analysed by presenting the information alongside the score. With an API, the score can be used interactively to feed information into the system. When such an API exists, complete systems which analyse musical performance issues can be created.

[1] Funded by the Arts & Humanities Research Council

Appendix A

Performance Markup Language Annotated DTD

```
<!-- ============================================================= -->
<!--                    Performance Markup Language DTD            -->
<!-- ============================================================= -->


<!--
        Performance Markup Language (PML) DTD
        Version 0.1 - 1 Oct 2005
        Copyright D McGilvray 2005 - 2006
        http://cmt.gla.ac.uk
                                                                    -->

<!--                    Include MusicXML part-wise DTD             -->
<!ENTITY % musicxml     SYSTEM 'partwise.dtd' > %musicxml;

<!-- ------------------------------------------------------------- -->
<!--
        Namespace prefix
        The namespace prefix defines the namespace prefix
        for all elements defined within the PML DTD. This
        can be redeclared to change the prefix of all elements
        to ensure compatibility with other namespaces.
```

```
                                                                    -->
<!ENTITY % nsp 'pml:' >

<!--
        Name entities
        The namespace prefix must be combined with a
        tag name within an entity. This also permits the
        renaming of elements' tag names if required.
                                                                    -->

<!ENTITY % n.pml           '%nsp;pml' >
<!ENTITY % n.performance   '%nsp;performance' >
<!ENTITY % n.perfpart      '%nsp;perfpart' >
<!ENTITY % n.event         '%nsp;event' >
<!ENTITY % n.onset         '%nsp;onset' >
<!ENTITY % n.offset        '%nsp;offset' >
<!ENTITY % n.midi          '%nsp;midi' >
<!ENTITY % atime           '%nsp;atime' >
<!ENTITY % n.freq          '%nsp;freq' >
<!ENTITY % n.align         '%nsp;align' >
<!ENTITY % n.wavres        '%nsp;wavres' >
<!ENTITY % n.wavref        '%nsp;wavref' >
<!ENTITY % n.tuning        '%nsp;tuning' >
<!ENTITY % n.refpitch      '%nsp;refpitch' >
<!ENTITY % n.midiref       '%nsp;midiref' >
<!ENTITY % n.rootpitch     '%nsp;rootpitch' >
<!ENTITY % n.intervallist  '%nsp;intervallist' >
<!ENTITY % n.interval      '%nsp;interval' >
<!ENTITY % n.spellinglist  '%nsp;spellinglist' >
<!ENTITY % n.enharmequiv   '%nsp;enharmequiv' >
<!ENTITY % n.pitch         '%nsp;pitch' >
<!ENTITY % n.pml           '%nsp;pml' >

<!-- ----------------- Basic Structure ----------------------------- -->

<!ELEMENT %n.pml;          (%n.score-partwise;,
                            %n.performance;)                        >
```

```
<!--    Contains Performance parts, external resources
        and tuning (if not 12ET)                                    -->
<!ELEMENT %n.performance;  (%n.tuning;?,
                            %n.extres;*,
                            %n.perfpart;*)                           >


<!-- Performance part                                               -->
<!-- Contains performance events relating to one score part.        -->
<!ELEMENT  %n.perfpart;     (%n.scorepartref*,
                             %n.event*)                              >
<!-- ----------------- Primitives ---------------------------------- -->

<!--                    Absolute time (seconds)                     -->
<!ELEMENT  %atime;        (#PCDATA)                                  >

<!--                    Midi number                                 -->
<!ELEMENT  %midi;         (#PCDATA)                                  >

<!--                    Frequency in Hertz                          -->
<!ELEMENT  %freq;         (#PCDATA)                                  >

<!--                    2DO: extend acc list                        -->
<!ENTITY % accidentals        '(Sharp | Flat | DoubleSharp | DoubleFlat)'>
<!ENTITY % pitchclasses        '(A|B|C|D|E|F|G) '                     >

<!--                    pitch                                       -->
<!ELEMENT  %n.pitch;       EMPTY                                     >
<!ATTLIST  %n.pitch;
             note       %pitchclasses;                    #REQUIRED
             acc        %accidentals;                    #IMPLIED
             octave     CDATA                           #REQUIRED    >



<!-- ------------------------------------------------------------- -->
```

```
<!--                    Event                                        -->
<!--                    Describes a performance artifact (Eg. a
                        note).                                       -->
<!ELEMENT  %n.event;    (%n.onset;,
                         %n.offset;,
                         %n.freq;?,
                         %n.pitch;?,
                         %n.midi;?
                         %n.align;?)                                  >
<!ATTLIST  %n.event;
             id         ID                              #REQUIRED    >


<!--    Links a performance event to a score item. The repeat
        attribute is used to distinguish between different
        occurrences of the same note in repeated segments            -->
<!ELEMENT  %n.align;        EMPTY                                     >
<!ATTLIST  %n.align;
             note       IDREF                           #REQUIRED    >
             repeat     CDATA                           #IMPLIED     >

<!--                    Onset and offset time in seconds or
                        using an external reference                  -->
<!ELEMENT  %n.onset      (#PCDATA?, %extref;)                         >
<!ELEMENT  %n.offset     (#PCDATA?, %extref;)                         >


<!-- ----------------Scope---------------------------------------- -->

<!--    The scope element provides a single, convenient method
        of defining a reference to one or more objects.
                                                                     -->

<!ELEMENT  %n.from;      (%atime;?, %extref;*)                        >
<!ATTLIST  %n.from;
               refs      IDREFS       #IMPLIED                      >

<!ELEMENT  %n.to;        (%atime;?, %extref;*)                        >
```

```
<!ATTLIST  %n.to;
                refs      IDREFS       #IMPLIED                    >

<!ELEMENT  %n.scope      (from, to)?                                 >
<!ATTLIST  %n.scope;
                refs      IDREFS       #IMPLIED                    >


<!-- ------------------External References------------------ -->

<!--  External references consist of a declaration of the resource
      including its location, an ID for referencing and
      default properties for referencing locations within the
      resource. These should be defined in the performance element.
      An external reference is used to reference a location in an
      external resource. It references the resource element by ID
      and contains an exact position within that resource
      Any implied properties defined in a resource element
      provide default values for all reference elements
      which reference that resource.
                                                                  -->

<!--  Parameter entity references for each external file type
      Each supported file type must have entries here         -->
<!ENTITY %      extres '(n.wavres | n.gmsres)'                      >
<!ENTITY %      extref '(n.wavref | n.gmsref)'                      >


<!--            WAV audio file                                  -->
<!ELEMENT       %n.wavres;          EMPTY                           >
<!ATTLIST       %n.wavres;
                file      ENTITY        #REQUIRED
                fps       CDATA               #REQUIRED
                channel   CDATA               #IMPLIED
>

<!ELEMENT       %n.wavref;          EMPTY                           >
<!ATTLIST       %n.wavref;
```

```
                wavres      IDREF          #REQUIRED
                channel     CDATA                 #IMPLIED
                frame       CDATA                 #REQUIRED
>

<!--                    GMS gesture file                              -->
<!ENTITY % n.gmsres '%nsp;gmsres'                                      >
<!ELEMENT       %n.gmsres;          EMPTY                              >
<!ATTLIST       %n.gmsres;
                file        ENTITY          #REQUIRED
                fps         CDATA                 #REQUIRED
                scene       CDATA                 #IMPLIED
                unit        CDATA                 #IMPLIED
                channel     CDATA                 #IMPLIED
                track       CDATA                 #IMPLIED
>

<!ENTITY % n.gmsref '%nsp;gmsref' >
<!ELEMENT       %n.gmsref;          EMPTY                              >
<!ATTLIST       %n.gmsref;
                gmsres      IDREF          #REQUIRED
                scene       CDATA                 #IMPLIED
                unit        CDATA                 #IMPLIED
                channel     CDATA                 #IMPLIED
                track       CDATA                 #IMPLIED
                frame       CDATA                 #REQUIRED
>


<!-- -------------- TUNING ------------------------------------- -->

<!--       Defined in the performance element. The ansence of any
           portion defaults to standard 12ET concert pitch i.e.
                        A4 = 440Hz
                        Midi 60 = C3
                        Root pitch = C
           and typical enharmonic equivalencies                    -->
```

```
<!ELEMENT  %n.tuning          (%refpitch;?,
                              %midiref;?,
                              %rootpitch?,
                              %intervallist;?,
                              %spellinglist;?)                          >
<!ATTLIST  %n.tuning;
             name        CDATA                                 #IMPLIED   >

<!--         midiref maps MIDI numbers to pitches                        -->
<!ELEMENT  %n.midiref;     (%n.pitch;, %n.midi;)                         >

<!--         refpitch maps a particular pitch to a frequency             -->
<!ELEMENT  %n.refpitch;    (%n.pitch;, %n.freq;)                         >


<!--         Root pitch assigns a pitch to be the first pitch
             in an interval list (i.e. the first interval in the
             interval list defines the interval between this pitch
             and the pitch next chromatic pitch. This has no effect
             on equal tempered tunings
                                                                         -->
<!ELEMENT  %n.rootpitch;   (%n.spelling;)                                >


<!--         The intervals within the tuning are defined as a list
             of intervals defined in either cents or an integer ratio
             The interval list is used to map frequencies on to the
             list of pitches and MIDI numbers using the rootpitch
             and refpitch definitions-->
<!ELEMENT  %n.intervallist; (%n.interval;+)                              >
<!ELEMENT  %n.interval;     (#PCDATA)                                    >

<!--         The notation is defined as a list of equivalent spellings.
             This is mapped to a list of integer pitches (like MIDI)
             using the midiref
                                                                        -->
<!ELEMENT  %n.spellinglist; (%n.enharmequiv;+)                           >
<!ELEMENT  %n.enharmequiv; (%n.spelling;+)                               >
```

```
<!--        A pitch spelling defines a pitch without an octave       -->
<!ELEMENT  %n.spelling;      EMPTY                                     >
<!ATTLIST  %n.spelling;
             note        %pitchclass;                    #REQUIRED
             acc         %accidental;                    #IMPLIED    >
```

Appendix B

Vocal Segmentation Examples

This appendix contains graphs which help to visualise the performance of the pitch algorithm described in chapter 4. Each graph represents a single note in the performance. The frequency trajectory is plotted as the deviation from 440Hz in cents against time in the x axis which is measured in frames (the frequency trajectory was analysed using a frame skip of 256 samples at 44100kHz sample rate). The blue line represents the initial, 'rough estimate' of the trajectory of the note. The final estimate is denoted by the green line. The 12 notes represent the section of the piece shown in figure 4.4 (excluding the final C) which occurs in bars 5 & 6.

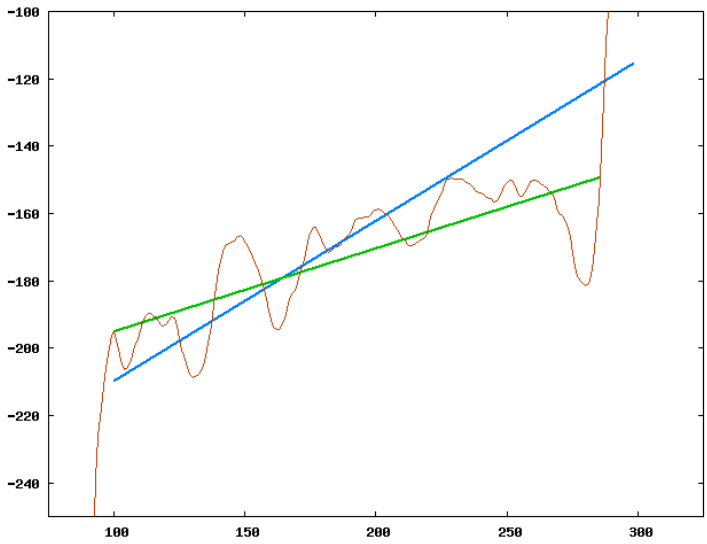


Figure B.1: Rough (blue) & final (green) estimate of the 1st note using the pitch trajectory.

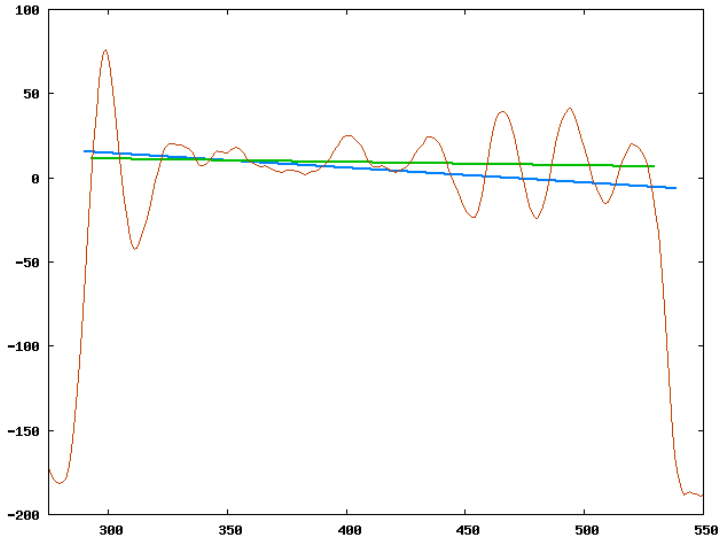


Figure B.2: Rough (blue) & final (green) estimate of the 2nd note using the pitch trajectory.

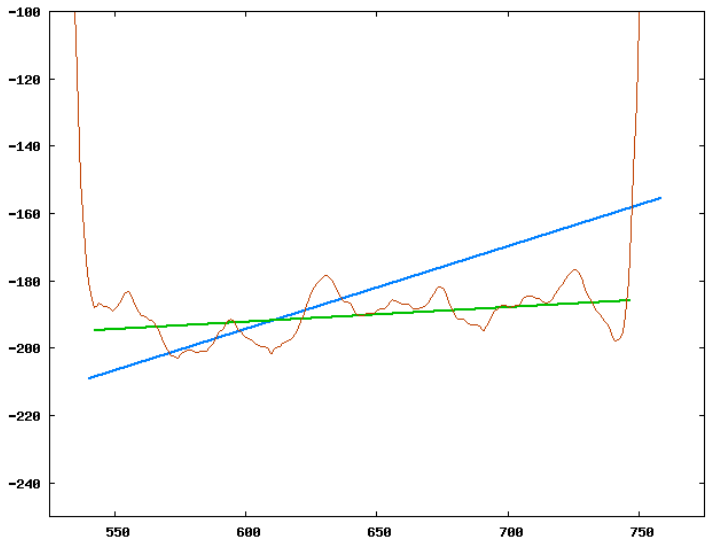


Figure B.3: Rough (blue) & final (green) estimate of the 3$^{\text{rd}}$ note using the pitch trajectory.

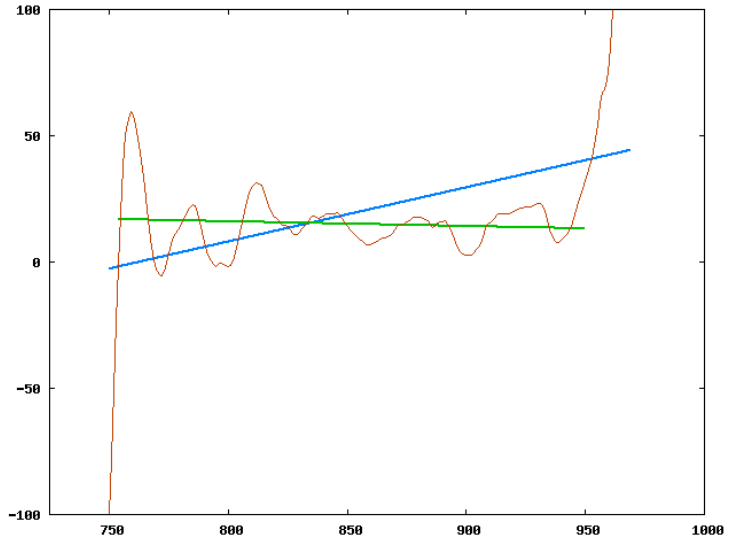


Figure B.4: Rough (blue) & final (green) estimate of the 4$^{\text{th}}$ note using the pitch trajectory.

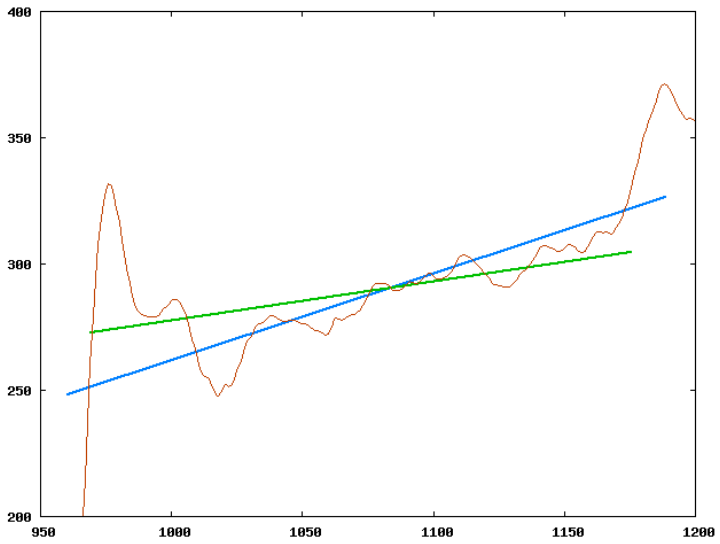


Figure B.5: Rough (blue) & final (green) estimate of the 5$^{\text{th}}$ note using the pitch trajectory.

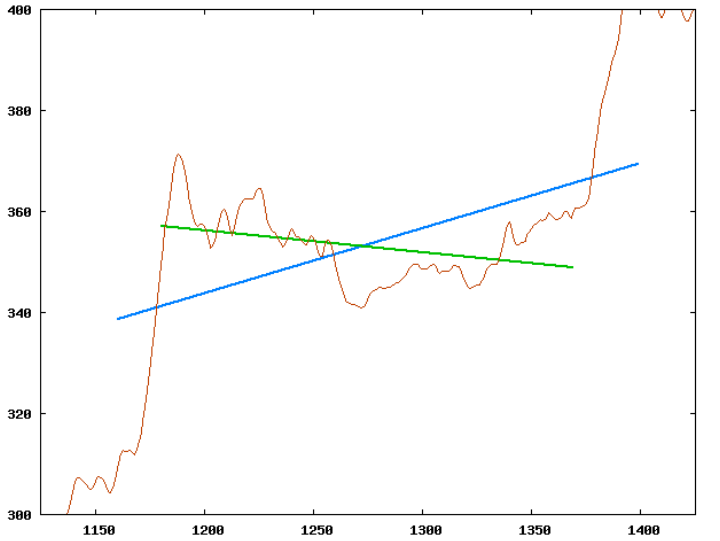


Figure B.6: Rough (blue) & final (green) estimate of the 6$^{\text{th}}$ note using the pitch trajectory.

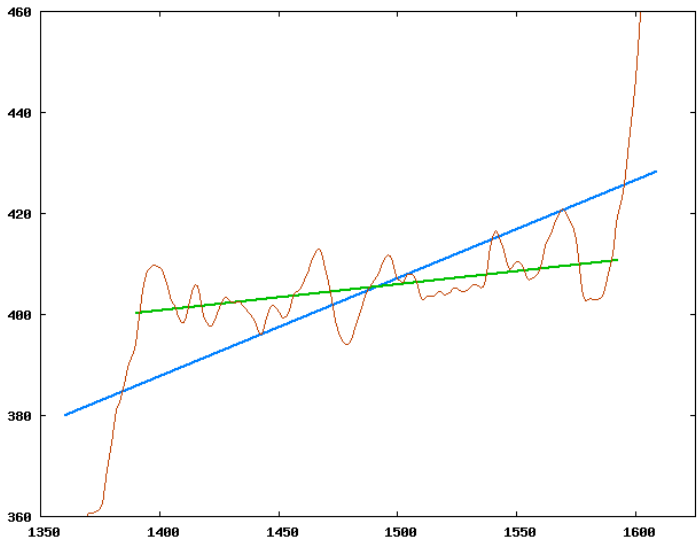


Figure B.7: Rough (blue) & final (green) estimate of the 7^{th} note using the pitch trajectory.

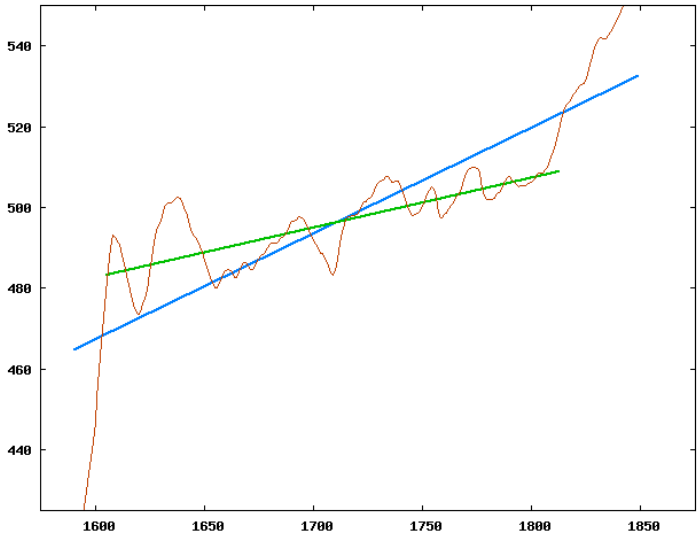


Figure B.8: Rough (blue) & final (green) estimate of the 8^{th} note using the pitch trajectory.

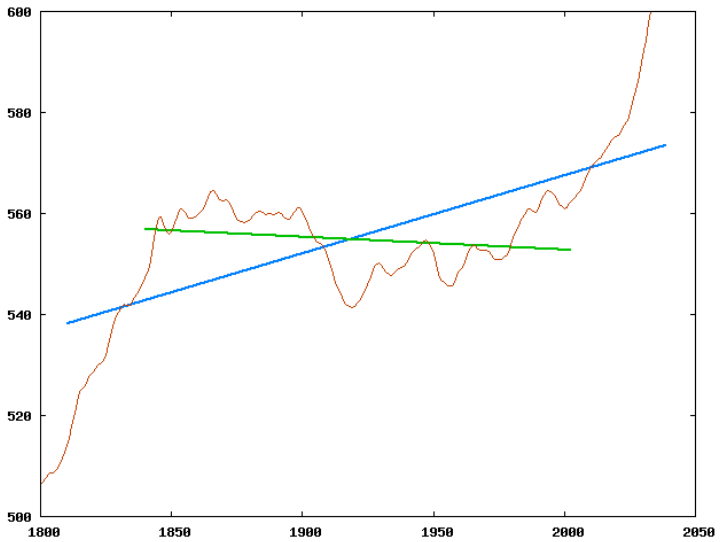


Figure B.9: Rough (blue) & final (green) estimate of the 9^{th} note using the pitch trajectory.

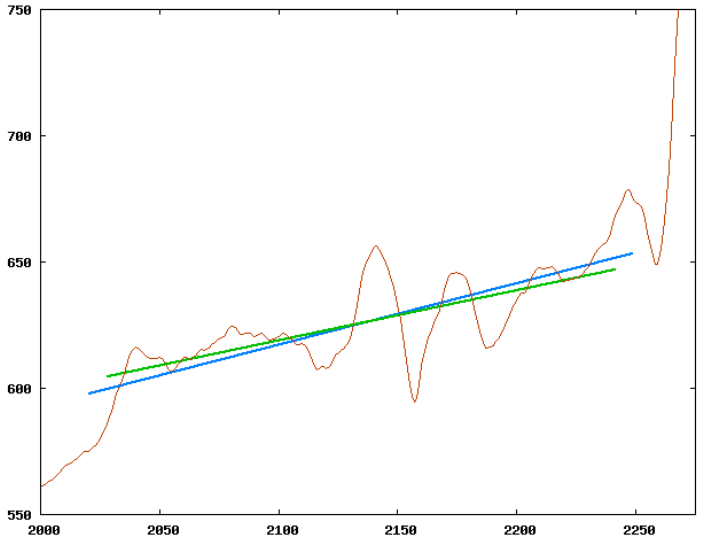


Figure B.10: Rough (blue) & final (green) estimate of the 10^{th} note using the pitch trajectory.

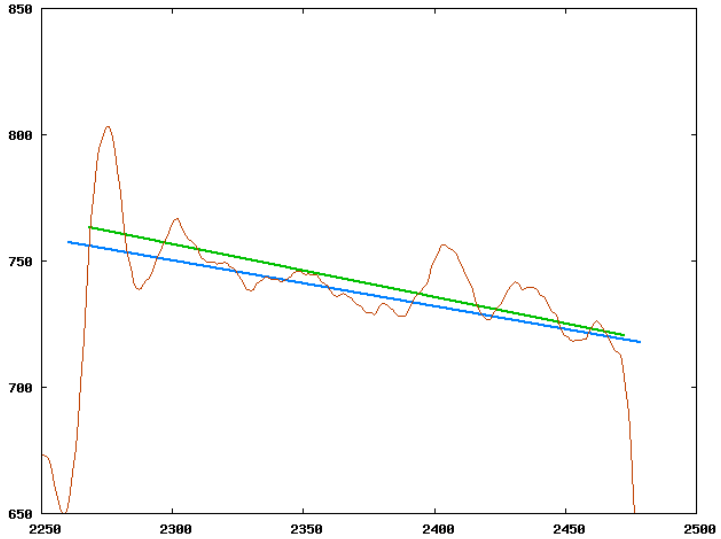


Figure B.11: Rough (blue) & final (green) estimate of the 11^{th} note using the pitch trajectory.

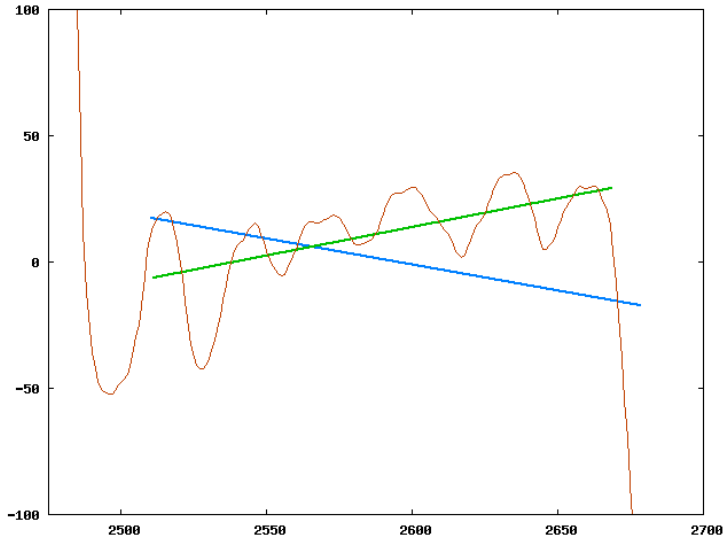


Figure B.12: Rough (blue) & final (green) estimate of the 12^{th} note using the pitch trajectory.

Bibliography

[1] Tellef Kvifte Alexander R. Jensenius, Rolf Inge Gody. Towards a gesture description interchange format. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2006.

[2] MIDI Manufacturers' Association. Complete midi 1.0 detailed specification v96.1 (second edition), 2001.

[3] Audacity [computer program]. Home Page: http://audacity.sourceforge.net/ (accessed 28 October 2007), 2007.

[4] J. Bello, G. Monti, and M. Sandler. Techniques for automatic music transcription. In *International Symposium on Music Information Retrieval*, 2000.

[5] Biovision. Bvh [file format]. http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html.

[6] Joshua J. Bloch and Roger B. Dannenberg. Real-time computer accompaniment of keyboard performances. In *ICMC*, pages 279–289, 1985.

[7] David Boersma, Paul & Weenink. Praat: doing phonetics by computer (version 4.6.12) [computer program]. Home Page: http://www.praat.org/ (accessed 28 October 2007).

[8] Jered Bolton. *Gestural Extraction from Musical Audio Signals*. PhD thesis, University of Glasgow, 2005.

[9] A. R. Brinkman. A binomial representation of pitch for computer processing of musical data. *Music Theory Spectrum*, 8:44–57, 1986.

[10] A. R. Brinkman. Representing musical scores for computer analysis. *Journal of Music Theory*, 1986.

[11] Chris Cannam. Sonic Visualiser [computer program]. Home Page: www.sonicvisualiser.org (accessed 28 October 2007), 2007.

[12] P. Cano, A. Loscos, and J. Bonada. Score-performance matching using HMMs. In *Proceedings of the ICMC*, pages 441–444, Beijing, 1999.

[13] Gerd Castan. Niffml: An xml implementation of the notation interchange file format. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *The Virtual Score*, chapter 7, pages 103–112. MIT Press, 2001.

[14] Gerd Castan, Michael Good, and Perry Roland. Extensible markup language (xml) for music applications: An introduction. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *The Virtual Score*, chapter 6, pages 95–102. MIT Press, Cambridge, MA, 2001.

[15] Cindy Grande (Tech. Coordinator). Niff Notation Interchange Format v.6a.3. Available at: http://neume.sourceforge.net/niff/ (accessed 28 October 2007), 1995.

[16] R. Dannenberg. Music understanding by computer. In *In IAKTA/LIST International Workshop on Knowledge Technology in the Arts*, pages 41–56, 1993.

[17] Roger B. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the ICMC*, 1984.

[18] Roger B. Dannenberg. The canon score language. *Computer Music Journal*, 1989.

[19] P. Desain and H. Honing. Tempo curves considered harmful. *Contemporary Music Review*, 7(2):123–138, 1993.

[20] Peter Desain and Henkjan Honing. Towards a calculus for expressive timing in music. *Computers in Music Research*, 3:43–120, 1991.

[21] Norbert Schnell Diemo Schwarz, Nicola Orio. Robust polyphonic midi score following with hidden markov models. In *Proceedings of the International Computer Music Conference (ICMC)*, Miami, Florida, 2004.

[22] Simon Dixon. An on-line time warping algorithm for tracking musical performances. In *International Joint Conference on Artificial Intelligence*, 2005.

[23] Mark Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[24] Martha Elliott. *Singing in Style: A Guide to Vocal Performance Practices*. Yale University Press, 2006.

[25] Matthieu Evrard, Damien Courouss, Nicolas Castagn, Claude Cadoz, Jean-Loup Florens, and Annie Luciani. A basic gesture and motion format for virtual reality multisensory applications. In *Proceedings of GRAPP conference*, 2006.

[26] International Organisation for Standardization. Information technology – hypermedia/time-based structuring language. ISO/IEC 10744:1997, 1997.

[27] Free Software Foundation, Inc. Gnu general public license [software license]. Available from http://www.gnu.org/, 2007.

[28] Michael Good. Musicxml: An internet-friendly format for sheet music. In *XML 2001 Conference Proceedings*, Orlando, FL, December 2001.

[29] Nelson Goodman. *Languages of Art*. Oxford University Press, London, 1969.

[30] Lorin Grubb and Roger B. Dannenberg. A stochastic method of tracking a vocal performer. In *ICMC*, 1997.

[31] Graham Hair, Ingrid Pearson, Amanda Morrison, Nicholas Bailey, Douglas McGilvray, and Richard Parncutt. The rosegarden codicil: Rehearsing music in nineteen-tone equal temperament. *Scottish Music Review*, 1(1), 2007.

[32] H. Heijink, Desain, H. P., Honing, and W. L. Windsor. Make me a match: An evaluation of different approaches to score-performance matching. *Computer Music Journal*, 2000.

[33] B. Hewlett, Walter. A base-40 number-line representation of musical pitch notation. *Musikometrika*, 4:1–14, 1992.

[34] Walter B. Hewlett. Musedata: Multipurpose representation. In *Beyond MIDI: The Handbook of Musical Codes*, chapter 27, pages 402–445. MIT Press, 1997.

[35] Keiji Hirata, Kenzi Noike, and Haruhiro Katayose. Proposal for a performance data format. In *Working Notes of IJCAI-03 Workshop on methods for automatic music performance and their applications in a public rendering contest*, 2003.

[36] Mr. G. Ken Holman. Secretariat manager's interim report, iso/iec jtc 1/sc 34, 23 May 2006.

[37] H. Honing. Poco: an environment for analysing, modifying, and generating expression in music., 1990.

[38] H. Hoos, K. Hamel, K. Renz, and J. Kilian. The guido notation format: A novel approach for adequately representing score-level music.

[39] H. Hoos, K. Hamel, K. Renz, and J. Kilian. Representing score-level music using the guido music-notation format. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *The Virtual Score*, chapter 5, pages 75–94. MIT Press, 2001.

[40] Takayuki Hoshishiba and Susumu Horioguchi. Improved DP matching between a musical score and its performance using interpolation. *Acoustical Science and Technology*, 22(1):13–19, 2001.

[41] J. Howard. Plaine and easie code: A code for music bibliography. In E. Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, chapter 25, pages 362–371. MIT Press, Cambridge Massachusetts, 1992.

[42] R. B. Hu, N. & Dannenberg. Polyphonic audio matching for score following and intelligent audio editors. In *Procceedings of the ICMC*, 2003.

[43] David Huron. A score-based study of musical dynamics in 14 piano composers. *Psychology of Music*, 19(1):33–45, 1991.

[44] David Huron. Humdrum and kern: Selective feature encoding. In E. Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, chapter 26, pages 375–398. MIT Press, 1997.

[45] Nicholas J. Bailey Jennifer MacRitchie and Graham Hair. Multi-modal acquisition of performance parameters for analysis of chopin's b flat minor piano sonata finale op.35. In *DMRN+1 Workshop, Digital Music Research Network*, 2006.

[46] A. R. Jensenius, A. Camurri, N. Castagn, E. Maestre, J. Malloch, D. McGilvray, D. Schwarz, and M. Wright. The need of formats for streaming and storing music-related movement and gesture data. In *Proceedings of the ICMC*, 2007.

[47] A. R. Jensenius, A. Camurri, N. Castagn, E. Maestre, J. Malloch, D. McGilvray, D. Schwarz, and M. Wright. A summary of formats for streaming and storing music-related movement and gesture data. In *Proceedings of the 4th International Conference on Enactive Interfaces (Accepted)*, 2007.

[48] Joseph B. Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM Review*, 25(2):201–237, April 1983.

[49] J. Langner and W. Goebl. Representing expressive performance in tempo-loudness space, 2002.

[50] E. W. Large. Dynamic programming for the analysis of serial behaviors. *Behavior Research Methods, Instruments, & Computers*, 25(2):238–241, 1993.

[51] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1965.

[52] Alan Marsden. *Representing Musical Time: A Temporal-Logic Approach*. Swets & Zeitlinger, 2000.

[53] R. McNab, L. Smith, and I. Witten. Signal processing for melody transcription. In *Proc. 19th Australasian Computer Science Conf.*, pages 301–307, 1996.

[54] Leonard B. Meyer. *Explaining Music*. University of California Press, 1973.

[55] F. Richard Moore. The dysfunctions of midi. *Computer Music Journal*, 12(1), 1988.

[56] Unjung Nam. Pitch distribution in korean court music: Evidence consistent with tonal hierarchies. *Music Perception*, 16(2):243–247, 1998.

[57] Tetsuo Sakata Masashi Yamamuro Kazuhiko Kushima Naoko Kosugi, Yuichi Nishihara. A practical query-by-humming system for a large

music database. In *Proceedings of the eighth ACM international conference on Multimedia*, pages 333 – 342, 2000.

[58] Dika Newlin. *Schoenberg Remembered: Diaries and Recollections*. Pendragon Press, New York, 1923.

[59] Han-Wen Nienhuys and Jan Nieuwenhuizen. Lilypond ... music notation for everyone. Home Page: http://lilypond.org/web/ (Accessed 28 October 2007), 2007.

[60] Roger B. Dannenberg Ning Hu and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003.

[61] Network for interdisciplinary studies in science, technology, and music [network]. Home Page: http://www.n-ism.org (accessed 28 October 2007).

[62] Jonah B. Shifrin Gregory H. Wakefield Norman H. Adams, Marj A Bartsch. Time series alignment for music information retrieval. In *Prceedings of ISMIR*, 2004.

[63] Noteedit v. 2.8.1 [computer program]. Home Page: http://noteedit.berlios.de/ (accessed 28 October 2007), 2007.

[64] Society of Motion Picture and Television Engineers. Smpte 12m-1999 television, audio & film - time and control code. Available at http://smpte.org (Accessed 28 October 2007), 1999.

[65] Uche Ogbuji. Principles of xml design: When the order of xml elements matters [web article]. Available at IBM Developer Works: http://www.ibm.com/developerworks/xml/library/x-eleord.html (Accessed 28 October 2007), April 2005.

[66] N Orio and F Déchelle. Score following using spectral analysis and hidden markov models. In *ICMC*, Havana, Cuba, 2001.

[67] Caroline Palmer. Anatomy of a performance: Sources of musical expression. *Music Perception*, 13(3):433–54, 1996.

[68] Caroline Palmer and Peter Q. Pfordresher. Incremental planning in sequence production. *Psychological Review*, 110(4):683–712, 2003.

[69] Bryan Pardo and William Birmingham. Modeling form for on-line following of musical performances. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005.

[70] Travis Pope, Stephen. Object-oriented music representation. *Organised Sound*, 1:56–58, 1996.

[71] R. E. Prather. Harmonic analysis from the computer representation of a musical score. In *Communications of the ACM*, volume 39, 1996.

[72] Ronald E. Prather and R. Stephen Elliott. Sml: A structured musical language. *Computers and the Humanities*, 22(2):137–151, 1988.

[73] M. Puckette. Score following using the sung voice. In *Proceedings of the ICMC*, 1995.

[74] Miller Puckette and Corte Lippe. Score following in practice. In *Proceedings of the ICMC*, pages 182–185, 1992.

[75] Ashwin Ram, Richard Catrambone, Mark J. Guzdial, Colleen M. Kehoe, D. Scott McCrickard, and John T. Stasko. Pml: Representing procedural domains for multimedia presentations. Technical report, GVU, Georgia Institute of Technology, Georgia Tech, Atlanta, Georgia, 1998.

[76] Christopher Raphael. Automatic segmentation of acoustic musical signals using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):360–370, 1999.

[77] Recordare. Capella professional [computer program]. Home Page: http://www.recordare.com/capella/capella.html Proceedings of the ICMC, 2007.

[78] Perry Roland. (Personal Communication dated: 30 May 2006).

[79] Perry Roland. The music encoding initiative (MEI) DTD and the OCVE. Available at: http://www.lib.virginia.edu/digital/resndev/mei/ (Accessed 28 October 2007).

[80] Perry Roland. Xml4mir: Extensible markup language for music information retrieval. In *ISMIR Proceedings*, 2000.

[81] Rosegarden sequencer [computer program]. Retrieved 31 September 2007: http://www.rosegardenmusic.com, 2007.

[82] Xavier Roset and Pierre Cointe. Formes: Composition and scheduling of processes. *Computer Music Journal*, pages 32–50, 1984.

[83] Rossignol S., Rodet X., Soumagne J., Collette J.-L., and Depalle P. Automatic characterisation of musical signals: Feature extraction and temporal segmentation. *Journal of New Music Research*, 28(4):281–295, 1999.

[84] W. Goebl S. Dixon and G. Widmer. The performance worm: Real time visualisation of expression based on langner's tempo-loudness animation.

[85] J. Soumagne J.-L. Collette P. Depalle S. Rossignol, X. Rodet. Feature extraction and temporal segmentation of acoustic signals. In *Proceedings of the ICMC*, 1998.

[86] J. Soumagne X. Rodet J.-L. Collette S. Rossignol, P. Depalle. Vibrato: Detection, estimation, extraction, modification. In *Proceedings Digital Audio Effects Workshop*, 1999.

[87] H. Sakoe and S. Chiba. Dynamic programming algorithm optimisation for spolen word recognition. In *IEEE Transactions on Acoustics, Speech and Signal processing*, volume 26, pages 43–49, 1978.

[88] E. Scheirer. *Music-Listening Systems*. PhD thesis, MIT Media Lab, 2000.

[89] B. Schottstaedt. Common music notation. In Eleanor Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, chapter 16, pages 217–221. MIT Press, 1997.

[90] M. R. Schroeder. Period histogram and product spectrum: New methods for fundamental-frequency measurement. *The Journal of the Acoustical Society of America*, 43(4):835–838, 1968.

[91] Second Hand Songs: a cover songs database. Homepage: http://www.secondhandsongs.com/, (Accessed 08 July 2007).

[92] Peter Seebach. Standards and specs: The interchange file format (iff) [web article]. Available at: IBM Developer Works http://www-128.ibm.com/developerworks/power/library/pa-spec16/?ca=dgr-lnxw07IFF, June 2006.

[93] E. Selfridge-Field. Darms, its dialects and its uses. In E. Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, chapter 11, pages 163–172. MIT Press, 1997.

[94] Eleanor Selfridge-Field. The musedata universe: A system of musical information. In *Computing in Musicology 9*, chapter 4. MIT Press, 1993.

[95] Eleanor Selfridge-Field. Describing musical information. In Eleanor Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, chapter 1, pages 3–37. MIT Press, 1997.

[96] Donald Sloan. Hytime and standard music description language. In E. Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, chapter 30, pages 469–489. MIT Press, 1997.

[97] Donald Sloan. Learning our lessons from smdl. In *MAX 2002 : musical application using XML*, 2002.

[98] Leland Smith. Score. In *Beyond MIDI: The Handbook of Musical Codes*, chapter 19, pages 252–282. MIT Press, 1997.

[99] Simon St.Laurent. *XML: Elements of Style*. McGraw-Hill, 2000.

[100] Motion Lab Systems. C3d [file format]. Home Page: http://www.c3d.org.

[101] Ichiro Fujinaga Takayuki Hoshishiba, Susumu Horiguchi. Study of expression and individuality in music performance using normative data derived from midi recordings of piano music. In *Proceedings of the 4th International Conference on Music Perception and Cognition*, pages 465–470, 1996.

[102] Semih Bilgen Uzay Bora, Selmin Tufan. A tool for comparison of piano performances. *Journal of New Music Research*, 29(1):85–99, March 2000.

[103] Caroline Palmer & Carla van de Sande. Units of knowledge in musical performance. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 19(2):457–470, 1993.

[104] Barry Vercoe. The synthetic performer in the context of live performance. In *In Proceedings of the ICMC*, pages 199–200, 1984.

[105] Barry Vercoe. *The CSound Reference Manual*. Addison-Wesley Publishing Company, 1991.

[106] W3C. Extensible markup language (xml) 1.0 (fourth edition).

[107] Chris Walshaw. abc music notation representation. Homepage at: http://www.walshaw.plus.com/abc/ Accessed 31 August 2007.

[108] M. M. Wanderley. Quantitative analysis of non-obvious performer gestures. In *Gesture and Sign Language in Human-Computer Interaction*, pages 241–253, 2002.

[109] Wedelmusic [computer program]. Home Page: http://www.wedelmusic.org/ (accessed 28 October 2007), 2007.

[110] C. Weihs and U Ligges. Automatic transcription of singing performances. In *Bulletin of the International Statistical Institute, 54th Session*, pages 507–510, 2003.

[111] Graham F. Welch, D. M. Howard, and C. Rush. Real-time visual feedback in the development of vocal pitch accuracy in singing. *Psychology of Music*, 17:146–157, 1989.

[112] G. Widmer. In search of the Horowitz factor: Interim report on a musical discovery project, 2002.

[113] Simon Dixon & Gerhard Widmer. Match: A music alignment tool chest. In *Proceedings of the 6th International Conference on Music Information Retrieval*, pages 492–497, 2005.

[114] G. Wiggins, E. Miranda, and M. Harris. Hierarchical music representation for composition and analysis. *Computing and the Humanities Journal*, 1993.

[115] G. Wiggins, E. Miranda, and Harris M. A framework for the evaluation of music representation systems. *Computer Music Journal, Vol. 17, No.3*, 1993.

[116] G. Wiggins, E. Miranda, and Harris M. Music representation - between the musician and the computer. In *World Conference on AI and Education workshop on Music Education*, 1993.

[117] Geraint A. Wiggins. Computer-representation of music in the research environment.

[118] Matthew Wright, Amar Chaudhary, Adrian Freed, and David Wessel. New applications of the sound description interchange format. In *Proceedings of the ICMC*, 1998.

[119] xml2ly [computer program]. Home Page:
http://www.nongnu.org/xml2ly/ (accessed 28 October 2007).