Linking Music Metadata

Robert Macrae

Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University of London.

School of Electronic Engineering and Computer Science
Queen Mary, University of London

March 12, 2012

I certify that this thesis, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline. I acknowledge the helpful guidance and support of my supervisor, Dr Simon Dixon.

Robert Macrae

Abstract

The internet has facilitated music metadata production and distribution on an unprecedented scale. A contributing factor of this data deluge is a change in the authorship of this data from the expert few to the untrained crowd. The resulting unordered flood of imperfect annotations provides challenges and opportunities in identifying accurate metadata and linking it to the music audio in order to provide a richer listening experience. We advocate novel adaptations of *Dynamic Programming* for music metadata synchronisation, ranking and comparison. This thesis introduces *Windowed Time Warping*, *Greedy*, *Constrained On-Line Time Warping* for synchronisation and the *Concurrence Factor* for automatically ranking metadata.

We begin by examining the availability of various music metadata on the web. We then review *Dynamic Programming* methods for aligning and comparing two source sequences whilst presenting novel, specialised adaptations for efficient, real-time synchronisation of music and metadata that make improvements in speed and accuracy over existing algorithms. The *Concurrence Factor*, which measures the degree in which an annotation of a song agrees with its peers, is proposed in order to utilise the wisdom of the crowds to establish a ranking system. This attribute uses a combination of the standard *Dynamic Programming* methods *Levenshtein Edit Distance*, *Dynamic Time Warping*, and *Longest Common Subsequence* to compare annotations.

We present a synchronisation application for applying the aforementioned methods as well as a tablature-parsing application for mining and analysing guitar tablatures from the web. We evaluate the *Concurrence Factor* as a ranking system on a large-scale collection of guitar tablatures and lyrics to show a correlation with accuracy that is superior to existing methods currently used in internet search engines, which are based on popularity and human ratings.

Acknowledgements

First and foremost I would like to thank my supervisor, Simon Dixon, for introducing me to Music Information Retrieval, guiding my work, and for his herculean effort in keeping me (mostly) focussed on the research relevant to this thesis. My other supervisors during this time included Mark Plumbley, who helped stress the importance of time management, Xavier Anguera Miro, who focussed me on the end product, and Joachim Neuman, who guided me on computational correctness and life correctness.

I would like to thank my examiners, Tim Crawford and Tijl De Bie for their excellent feedback as well as my colleagues: Matthias Mauch, Amélie Anglade, Dan Stowell, Andrew Robertson, Adam Stark, Mathieu Barthet, Enrique Perez Gonzalez, Gyorgy Fazekas, Sefki Kolozali, and Andrew Nesbit for their exemplary work, feedback, and discussion. I am indebted to Chris Harte, Chris Cannam, Craig Stuart Sapp, Markus Schedl, and Dan Ellis, for their work has made much of this possible.

I am also grateful for Tom, Martin, Tony, and Byron, for ensuring I took occasional time out. Finally I would like to thank my family for their support, an extra mention going to Andi and Steve for proof reading work in this thesis, and Jana, for her encouragement, her patience, and her faith this too shall pass.

This work was supported financially by a Doctoral Training Account from the Engineering and Physical Sciences Research Council and a travel grant from the Royal Engineering Society.

Contents

List of Figures

List of Tables

1 Introduction

"It will go down in history as a turning point for the music industry." Steve Jobs[1]

Contents

Our interaction with music is evolving. Personal collections filling hardware at home or in the pocket are giving way to unlimited cloud services and music streaming delivered through novel interfaces and applications. We are no longer simply consumers of music, instead we provide our own content in the form of ratings, reviews, likes, and tags, whilst the internet and social networks allow us to create and share playlists, mixes and annotations. Whereas traditionally music annotations were the domain of a professional minority, the requirements for authoring annotations on the web are much less restrictive, and as a result the web provides a larger collection of imperfect representations reflecting the typical user's understanding of the music. At the time of writing, we are entering a period commonly referred to as the "Data Deluge" or "Petabyte Age" [Baraniuk, 2011]. There was more data transmitted over the Internet in 2010 than the entire history of the Internet through 2009 [Skaugen, 2011] and every minute, two days worth of video is posted to YouTube [Richmond, 2011]. Analysing, organising and linking music and metadata from this data flood brings new possibilities and challenges for research in Music Information Retrieval (MIR) [Seltzer and Zhang, 2009].

[1] http://arts.guardian.co.uk/netmusic/story/0,13368,1241746,00.html

This thesis is concerned with automatically finding, analysing and linking large-scale music metadata with music audio. The volume of metadata available on the web and the variety of its sources require new MIR methods that can establish these connections and scale efficiently. Our work focuses on making accurate music metadata more accessible. In order to achieve this we explore methods that mine the web for various metadata, ranking algorithms to sort this metadata, and techniques for aligning metadata and judging its quality.

1.1 Music Metadata Alignment

There is a large amount of information that is chronologically relevant to music recordings. This metadata can be in the form of descriptions, musical scores, lyrics, beats, tags, onsets, features, gestures, movements, or other audio. Music metadata synchronisation involves the linking of audio and its metadata so that the events in one source can be related to those in the other. In practice, this involves taking two separate, albeit related, sources of audio and metadata then mapping the events or features of one to the other so that equivalent parts of both sources are related to one another regardless of differences in length, structure or tempo. This can either be performed in real-time, where one of the streams is being received live, or off-line, where both sources are already known completely. This process is often referred to as synchronisation or alignment.

Synchronisation has a wide range of applications within the field of digital music. Off-line alignment can be used to link sources in a database [Keogh and Pazzani, 2000], attach lyrics to song files to be used as Karaoke data [Kan et al., 2008; Wang et al., 2004], and to create annotated test data for other Digital Music methods such as automatic transcription [Turetsky and Ellis, 2003]. Real-time synchronisation allows for music that adjusts to users' gestures or other musicians changing rhythm [Robertson and Plumbley, 2007], and automatic score following which in turn can drive page turners or automatic accompaniment [Dannenberg, 1984; Raphael, 2001]. Music or music videos can also be kept in synchronisation with the listeners' movements as demonstrated in Cati Dance [Jehan et al., 2003]. If a piece of audio is aligned and linked to its musical score then you can navigate a piece of music by searching for features, segments of music [Viro, 2011] or lyrics [Kan et al., 2008].

Dynamic Programming methods have become the standard means of synchronising two sequences. Dynamic Programming is a term encompassing a wide range of problem solving techniques that break tasks down into sequentially smaller problems. A common use of Dynamic Programming methods is to find optimal route or best fit solutions in a manner that is guaranteed to find the optimal path. One such method, Dynamic Time Warping, uses Dynamic Programming to find the minimal cost alignment through a similarity matrix of two source sequences [Soulez et al., 2003; Turetsky and Ellis, 2003; Arifi et al., 2004]. Dynamic Time Warping was first used to synchronise audio in the context of word recognition in the 1970's [Itakura, 1975; Sakoe and Chiba, 1978; Myers et al., 1980; C. Myers and L. Rabiner and A. Rosenberg, 1980]. Since then it has been applied to other types of speech processing, audio-audio alignment [Dixon, 2005], data mining [Keogh and Pazzani, 2000], gesture recognition [Müller, 2007], face recognition [Bhanu and Zhou, 2004], medicine [Vullings et al., 1998], analytical chemistry [Clifford et al., 2009], and other areas. Synchronisation tasks that are required to work in real-time are typically implemented using another category of Dynamic Programming methods called probabilistic models [Cano et al., 1999; Raphael, 2001; Schwarz et al., 2004; Cont and Schwarz, 2006] that estimate sequence alignments but these probabilistic models require a set of parameters to be either chosen by hand or trained on suitable data.

Dynamic Time Warping based alignment techniques are limited by three major drawbacks. Firstly, these methods require the complete sequences of both sources, ruling out real-time synchronisation of live music [Dixon, 2005]. Secondly, these techniques assume the alignment is a singular and complete one-to-one relationship between the source sequences that starts and ends at the beginning and end of both sequences. Finally, aligning large sequences becomes computationally expensive in memory and processing requirements as Dynamic Time Warping techniques often have quadratic time and space costs. Although efficient [Salvador and Chan, 2004] and real-time [Dixon, 2005] modifications of Dynamic Time Warping exist and have been applied to music synchronisation, these improvements are limited to ensuring linear scaling of the processing time required. In order to link music metadata on large, web-scale resources and in data mining [Ratanamahatana and Keogh, 2004], further efficient synchronisation techniques need to be addressed. This leads to our first research question (RQ1):

RQ1: Can we improve the efficiency of current Dynamic Time Warping approaches to synchronising music and metadata without significant loss in accuracy?

Any improvement that makes Dynamic Time Warping methods more efficient in scaling will likely result in a loss of the guaranteed optimal solution [Myers et al., 1980]. As such, there is a requirement to measure the accuracy trade off of any modifications. However, assessing synchronisation techniques is typically problematic, as producing ground truth data for automatic alignment methods is a difficult and time consuming procedure. Additionally, as automatically aligned data can be used as ground truth or training data for other MIR methods [Turetsky and Ellis, 2003; You and Dannenberg, 2007], there is a need to compare machine produced data with manual data. As such, research question two (RQ2) is:

RQ2: How can we judge the capabilities of music metadata synchronisation methods and can machine produced ground truth data provide a reliable indication of real-world accuracy to the same degree that manually produced data can?

Linking the increasing number of different music modalities has been the focus of recent research in MIR [Müller et al., 2011] as the web has enabled new modalities of metadata based on text and web-content mining [Liu, 2006]. A common method for analysing web content is to use text mining/content analysis techniques [Neuendorf, 2002] and a summary of applicable MIR web metadata mining methods can be found in [Schedl, 2008]. Our next research objective is to examine the possibility of linking one type of metadata, that of guitar tablatures and chord sequences, that have been largely unexplored in MIR [McVicar et al., 2011b]. With over 4.5 million tablatures and chord sequences (which we collectively refer to as tabs), the web holds vast quantities of annotations of music in non-standardised text files and these tabs have become the most popular method of sharing annotations of music online. However, these annotations are typically error-prone and incomplete, and tab collections contain many duplicates and versions of the same files, making retrieval of high quality tabs a challenge. Due to the lack of a standard format for guitar tabs, no tools exist to interpret guitar tabs for music information retrieval and no metrics exist to judge their accuracy. Our next research question (RQ3) is therefore:

RQ3: Can text-mining techniques be used to reliably interpret non-standard and noisy guitar tabs and, using these techniques, can we evaluate their accuracy?

Guitar tabs and song lyrics are two types of music annotation that exist in large quantities on the internet. However, as there are no restrictions in publishing this material, many are annotated incorrectly and finding the accurate annotations can be challenging. Typical methods for ranking metadata have shown little correlation between the rank and the accuracy of the metadata [Macrae and Dixon, 2011]. We examine means of evaluating the accuracy of metadata and ranking them using Dynamic Programming so that the most relevant metadata can be identified in research question (RQ4):

RQ4: Is it possible to apply Dynamic Programming to judge the accuracy of metadata and identify the most relevant music annotations from large scale collections?

In this thesis, we aim to answer the aforementioned research questions in our bid to make music metadata more accessible and develop tools for automatically finding, analysing and linking large-scale music metadata with music audio.

1.2 Contributions

In this thesis, we present the following modifications of Dynamic Programming; the *Greedy* method (3.2), the *Two-Step* algorithm (3.3), *Windowed Time Warping* (3.4), *On-Line Time Warping Jumping* (3.5), and *On-Line Time Warping Constrained* (3.6), as well as extensions to these methods. Additionally, a *Quick Start* (3.7) method is proposed that can be applied to any of these Dynamic Programming modifications to find an alternative alignment path. The *Concurrence Factor* (6.1.6) is proposed in order to utilise the wisdom of the crowds to establish a ranking system and measures the degree in which an annotation of a song agrees with its peers. This attribute uses a combination of the standard *Dynamic Programming* methods *Levenshtein Edit Distance*, *Dynamic Time Warping*, and *Longest Common Subsequence* to compare annotations' similarity. As well as these methods, we present applications for deploying synchronisation algorithms (3.8), for authoring test data (4.1.1), for web mining guitar

tabs and interpreting guitar tabs (5.6). We evaluate real-time modifications of Dynamic Programming (Chapter 4) and guitar tab/song lyrics ranking algorithms (Chapter 6). Finally we advocate using semi-supervised machine-produced ground truth data in MIR evaluations and annotations, new Dynamic Programming modifications for large scale synchronisation tasks, and the *Concurrence Factor* for ranking metadata and reliably indicating accuracy in large noisy datasets in Chapter 7.

1.2.1 Publications

The following publications contain work that is incorporated in this thesis.

Papers

Robert Macrae and Simon Dixon, "From toy to tutor: Notescroller is a game to teach music" In proceedings of the International Conference on New Interfaces for Musical Expression (NIME) [Macrae and Dixon, 2008]

Robert Macrae and Simon Dixon, "A Guitar Tablature Score Follower" In proceedings of the IEEE International Conference on Multimedia and Expo (ICME) [Macrae and Dixon, 2010a]

Robert Macrae and Simon Dixon, "Accurate Real-time Windowed Time Warping" In proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR) [Macrae and Dixon, 2010b]

Robert Macrae and Simon Dixon, "Guitar Tab Mining, Analysis and Ranking" In proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR) [Macrae and Dixon, 2011]

Mathieu Barthet, Amélie Anglade, Gyorgy Fazekas, Sefki Kolozali, Robert Macrae, "Music recommendation for music learning: Hotttabs, a multimedia guitar tutor" In proceedings of the 2nd Workshop On Music Recommendation And Discovery (WOMRAD) [Barthet et al., 2011]

Under Review

Robert Macrae and Simon Dixon, "Evaluating Real-time Dynamic Time Warping methods for Score Following", submitted to IEEE Transactions on Audio, Speech, and Language Processing.

Other Publications

Robert Macrae, Xavier Anguera, and Nuria Oliver, "MuViSync: Realtime Music Video Alignment" Proceedings of the IEEE International Conference on Multimedia and Expo (ICME) [Macrae et al., 2010]
Robert Macrae, Joachim Neumann, Xavier Anguera, Nuria Oliver, and Simon Dixon "Real-Time Synchronisation of Multimedia Streams in a Mobile Device" Proceedings of the IEEE International Conference on Multimedia and Expo (ICME) [Macrae et al., 2011]

1.3 Thesis Structure

The remainder of this thesis is structured as follows. In Chapter 2, we present related work relevant to this research and review the availability of different music metadata on the web. In Chapter 3, to answer research question RQ1, we present Dynamic Programming modifications for improving music metadata synchronisation in real-time, including *Windowed Time Warping*, *Greedy*, and *Constrained On-Line Time Warping*, and other methods. In Chapter 4 we present tools for authoring score following or alignment test-data and an evaluation of real-time DTW modifications aimed at resolving RQ2. In Chapter 5 we detail our experiments in web mining guitar tablature and chord sequences, present a guitar tab parser based on text-mining, and evaluate the accuracy of online tabs to answer RQ3. In Chapter 6 we introduce the *Concurrence Factor* and evaluate annotation ranking methods on web mined guitar tabs and song lyrics in order to address RQ4. Finally, in Chapter 7 we summarise this research and discuss future topics highlighted by this work.

2
Preliminaries

"In theory, theory and practice are the same. In practice, they are not." Albert Einstein

Contents

In this chapter we introduce the research areas and review work related to the aims of this thesis in order to establish the framework upon which the rest of this work is based. We begin by looking at the various types of music metadata and what is contributing to their growing availability in Section 2.1. We then establish the Dynamic Programming algorithms that we will use later on in this work, and their role in linking music and metadata, in Section 2.2. We include an overview of research trends in Dynamic Programming within MIR in Section 2.3, and outline the potential applications and expected outcomes of research in linking music and metadata in Section 2.4. We then draw conclusions on the related research and work in linking music and metadata in Section 2.5.

2.1 Music Metadata

Music metadata, or data about music, can encompass many types of music related information. Typically, the music industry refers to metadata as the textual descriptions of a particular

recording or album such as the name of the artist or track, genre, key, tempo, duration, *etc.* Other textual descriptions of music come in the form of labels, reviews, links, user ratings, and comments applied to the music. Music annotations are textual or graphical descriptions of the underlying music score or instructions of how to reproduce it. Common examples of annotations are standard sheet music, lyrics, tablature, piano rolls, Musical Instrument Digital Interface (MIDI) files and the increasingly popular scrolling displays in games such as Guitar Hero[1] and Rock Band.[2] Features are a type of metadata that are audio descriptors of a song, typically obtained by analysing the audio, such as beats per minute, pitch, frequency, energy, and zero crossing rate. We also consider related media, such as cover art, music videos and music events as metadata.

Every type of music metadata described is becoming more readily available as part of the data deluge. Web Content, Social Networks, Games With A Purpose, Music API's, and Cloud Services are all contributing factors leading to growth in music metadata production and are all based on one underlying technology, the internet. Each of these sources of data has proven to be exploitable for research in MIR and here we give a brief overview of MIR in these areas.

2.1.1 Web Mining

There are various sources of web content. Band and fan websites are becoming increasingly common as artists are mastering the internet to gain exposure. The textual information in these web sites, as well as their structural links with other artists, can be crawled using text mining techniques. Web mining is typically divided into three typical categories: Web usage mining, Web content mining, and Web structure mining. Web usage mining involves analysing users' actions and behavioural patterns, such as examining web logs or commonly searched terms. A typical example of web usage mining would be the use of Collaborative Filtering (CF) on user generated lists. CF is the use of co-occurrences within user preference metadata for purposes such as recommendation. For example, CF using co-occurrences of artists and tracks has been shown to indicate their similarity [Pachet et al., 2001] within compilation discs and radio playlists in a database of 5000 tracks. See Su and Khoshgoftaar [2009] and Borchers et al. [1998] for an overview of CF techniques. Baumann and Hummel [2005] examine "cultural

[1] http://hub.guitarhero.com/
[2] http://www.rockband.com/

metadata" gleaned from web usage mining for music recommendation. Similarly, Zadel and Fujinaga [2004] mine cultural metadata in the form of artist co-occurences from Google and Amazon web services to establish artist similarity. Web content mining is the analysis of text and media content within websites. zu Eissen and Stein [2004] analyse web pages for genre classification. Furthermore, Schedl et al. [2008] demonstrate a Music Information System for providing multimodal information such as artist, track text descriptors, biographies, song samples, and album covers which gathers all its data through web content mining. Web structure mining examines the links between web resources. Web crawlers, ranking algorithms, and search engines such as Google [Page et al., 1999] typically mine web structure information. Fields et al. [2008] and Jacobson et al. [2008] examine social network connections for audio/artist similarity. For an in depth review of data mining techniques and how they apply to the semistructured and unstructured nature of the web, we refer the reader to Liu [2006].

2.1.2 Lyrics

Multiple copies of song lyrics are usually available on the internet for any song. As lyrics are relatively easy to mine from the web, and given that the words to a song contain rich semantic information, lyric information retrieval is a key area of interest within MIR. A contributing factor to the abundance of lyrics and a potential problem for research in this area is the lack of requirements, such as training or language knowledge, that are typically necessary for professionally annotating lyrics. Due to these issues, there is a high potential for song lyrics to contain errors. Knees et al. [2005] and Korst and Geleijnse [2006] use multiple sequence alignment to automatically retrieve accurate lyrics by linking lyrics of the same song. Lyrics and music are automatically synchronised using Dynamic Programming in LyricAlly [Wang et al., 2004; Kan et al., 2008] and other systems [Wong et al., 2007; Fujihara et al., 2006; Mauch et al., 2010] for the purposes of karaoke, song-browsing and thumbnailing. The content of a song's lyrics could also indicate musical structure and the language used could imply which genre it belongs to [Mayer et al., 2008] or the topic of the song [Kleedorfer et al., 2008]. Logan et al. [2004] use text processing to mine the semantic content of lyrics for music indexing, artist similarity, and genre classification. Another example of lyric based information retrieval uses natural language

processing to extract language, structure, categorisation, and similarity from lyrics in Mahedero et al. [2005].

2.1.3 Guitar Tablature and Chord Sequences

Guitar tablatures and chord sequences are popular and noisy sources of music annotations on the web. Both tablatures and chord sequences can commonly be found on guitar "tab" websites. Although the most common format of musical score on the web, guitar tablature has been largely ignored in MIR research, possibly because these "tabs" contain a large number of errors compared to professionally annotated scores, have no standard format, are presented in a way that is not trivial for a machine to parse, and there exist no reliable means to distinguish which are accurate [Macrae and Dixon, 2011]. Alternatively, chord sequences have been used in recent years within MIR techniques. Mauch et al. [2010] use chord sequences for aligning audio and lyrics. García-Díez et al. [2011] use chord sequences for cover song detection. Chord sequences have also been used to improve Dynamic Programming methods for automatic chord recognition [McVicar et al., 2011a,b], and to infer meaning to the chord sequences from lyrics [O'Hara, 2011].

2.1.4 Social Networks

Social Networks such as MySpace[3] and Facebook[4] allow users to share musical preferences or even what they are listening to at any given moment. Additionally, artists have embraced Social Networks as marketing tools to connect to fans and have set up band profiles with links to associated artists. The links between artists have been found to be indicators of audio/artist similarity and genre using web structure mining within Myspace using a subset of 15478 Myspace links [Fields et al., 2008; Jacobson et al., 2008]. Fields et al. [2011] use this metadata for music recommendation using standard complex network techniques and graph theory. This source of metadata is growing with the new partnership between Spotify[5] and Facebook providing links between artists and 400 million playlists.[6]

[3] http://www.myspace.com/
[4] http://www.facebook.com/
[5] http://www.spotify.com/
[6] http://www.spotify.com/uk/blog/archives/2011/09/21/spotify-and-facebook/

2.1.5 Games With A Purpose

Luis von Ahn introduced Games With A Purpose (GWAP) with the "Completely Automated Public Turing test to tell Computers and Humans Apart" or CAPTCHA system [von Ahn et al., 2003] and image labelling game [von Ahn and Dabbish, 2004]. In GWAP, players provide solutions to a problem simply by playing a game. With CAPTCHA [von Ahn et al., 2003], this involves typing a word that is obscure in order to provide access to a web service, with the results being compared against other users' input to see if the two agree. The obscure word is chosen because a machine can not recognise it. There are two benefits to this system: the first is that the web service can distinguish between real users and software agents, and the second is that a previously unsolvable AI problem has just been solved willingly by two humans agreeing on how the word should be interpreted. The image labelling game, called the ESP Game [von Ahn and Dabbish, 2004], involves two separated players attempting to agree labels for images. Although no reward was offered to users other than the joy of participating in the game and collecting points, players would participate and tag images in a way computers could not. By 2006, the ESP Game was responsible for over 31 million image tags[7] and to date, 750 million users have helped transcribe at least one CAPTCHA.[8] Another game, Phylo,[9] involves players solving multiple sequence alignments to align nucleotide sequences.

Within MIR, there have been a number of GWAP for gathering music metadata. TagATune [Law et al., 2007] and MajorMiner [Mandel and Ellis, 2008] both follow the ESP Game formula, using audio clips in place of the images. HerdIt [Barrington et al., 2009] is a Facebook application with more simultaneous players providing tags in response to specific challenges, such as selecting the correct genre. The tags produced by these games contain rich semantic information that can be used to help music recommendation or categorisation.

2.1.6 Application Programming Interface

An Application Programming Interface (API) is a set of rules and specifications that allow software programs and web services to communicate autonomously. APIs have allowed for direct access to metadata collections in music services, creating opportunities for music information

[7] http://www.gwap.com/gwap/gamesPreview/espgame/
[8] http://vonahn.blogspot.com/2010/07/work-and-internet.html
[9] http://phylo.cs.mcgill.ca/

retrieval on large commercial datasets. The Programmable Web[10] lists 129 music related APIs. These APIs and the metadata they give access to include MusicBrainz,[11] The Echo Nest,[12] and MusicMetric[13] for textual music descriptors, 8tracks[14] and Last.fm[15] for user generated playlists, and Songkick[16] and Live Nation[17] for metadata concerning live music events. The YouTube API[18] is used in MusicPlanner [Yang, 2010] for music playlist generation and recommendation. Tingle et al. [2010] link music with tags from Pandora's Music Genome Project[19] using audio features from The Echo Nest.

For a more in depth overview of music metadata mining and linking, Mayer and Rauber [2010] review recent developments in multimodal music information retrieval with a focus on audio and lyrics analysis.

2.2 Dynamic Programming in MIR

Dynamic Programming (DP) is the use of recursive computation to solve complex mathematical problems by breaking them down into repeatedly smaller subproblems. DP was originally formularised by Richard Bellman in the 1940s and 1950s [Bellman, 1957] and, since the advent of computers, has rapidly grown into an umbrella group of algorithms for solving widespread complex problems. DP is now used in fields as diverse as economics [Beckmann, 1968], artificial intelligence [Ouyang and Shahidehpour, 1992], computer graphics [Ishimura et al., 1986], medicine [Vullings et al., 1998], analytical chemistry [Clifford et al., 2009] and natural language processing [Strube et al., 2002].

Although common in speech recognition since the 1970s [Itakura, 1975; Sakoe and Chiba, 1978], DP was first used within the context of MIR in Dannenberg's real-time accompaniment score follower [Dannenberg, 1984] in 1984. The growth of DP as a sequence alignment tool

[10] http://www.programmableweb.com
[11] http://musicbrainz.org/
[12] http://the.echonest.com/
[13] http://www.musicmetric.com/
[14] http://8tracks.com/
[15] http://www.last.fm/
[16] http://www.songkick.com/
[17] http://www.livenation.co.uk/
[18] http://www.youtube.com/
[19] http://www.pandora.com/mgp.shtml

and similarity measure in MIR since then is examined in Section 2.3, whereas here we give an overview of the various DP based methods that will be referred to within this thesis.

2.2.1 Levenshtein Edit Distance

The Levenshtein Edit Distance (LED) [Levenshtein, 1966] counts the number of "edits" required to transform one string into another. An edit is classed as an insertion, deletion, or substitution of a single character. LED uses a cost of 0 for matches and 1 for any edit (insertion, deletion or alteration). As such the LED of "sun" and "sing" is 2 (substitution of the letter 'u' for 'i' and insertion of the letter 'g'). The LED cost is found by calculating a path $P(U,V) = (p_1, p_2, ..., p_W)$ through a matrix of costs between strings $U = (u_1, u_2, ..., u_M)$ and $V = (v_1, v_2, ..., v_N)$. This cost matrix is described as $d_{U,V}(m,n)$ where $m \in [1:M]$ and $n \in [1:N]$ where each $p_k = (m_k, n_k)$. A simple bottom-up algorithm for calculating the LED in $O(N^2)$ time and space is shown in Algorithm 1. In this example a matrix of edit costs is calculated between two strings, so that the cell in the final row and column would contain the total number of required edits. Additionally, an example of the "cost matrix" and the solution this algorithm produces can be seen in Table 2.1.

Input: String A, String B
Output: Levenshtein Edit Distance LED
Matrix m; $m[0,0] := (A[0] == B[0]?\ 0 : 1)$;
for $a \in [1..A.length]$ **do**
 | $m[a,0] := (A[a] == B[0]?\ 0 : 1) + m[a-1,0]$;
end
for $b \in [1..B.length]$ **do**
 | $m[0,b] := (B[b] == A[0]?\ 0 : 1) + m[0,b-1]$;
end
for $a \in [1..A.length]$ **do**
 | **for** $b \in [1..B.length]$ **do**
 | | $m[a,b] := (A[a] == B[b]?\ m[a-1,b-1] :$
 | | $1 + min(m[a-1,b], m[a-1,b-1], m[a,b-1]))$;
 | **end**
end
return $LED := m[A.length, B.length]$;

Algorithm 1: The Levenshtein Edit Distance.

String A: you are the music
String B: while the music lasts

	y	o	u	a	r	e	t	h	e	m	u	s	i	c
w	**1**	**2**	3	4	5	6	7	8	9	10	11	12	13	14
h	2	2	**3**	4	5	6	7	7	8	9	10	11	12	13
i	3	3	3	**4**	5	6	7	8	8	9	10	11	11	12
l	4	4	4	4	**5**	6	7	8	9	9	10	11	12	12
e	5	5	5	5	5	**5**	6	7	7	8	9	10	11	12
t	6	6	6	6	6	6	**5**	6	7	8	9	10	11	12
h	7	7	7	7	7	7	6	**5**	6	7	8	9	10	11
e	8	8	8	8	8	7	7	6	**5**	6	7	8	9	10
m	9	9	9	9	9	8	8	7	6	**5**	6	7	8	9
u	10	10	9	10	10	9	9	8	7	6	**5**	6	7	8
s	11	11	10	10	11	10	10	9	8	7	6	**5**	6	7
i	12	12	11	11	11	11	11	10	9	8	7	6	**5**	6
c	13	13	12	12	12	12	12	11	10	9	8	7	6	**5**
l	14	14	13	13	13	13	13	12	11	10	9	8	7	**6**
a	15	15	14	13	14	14	14	13	12	11	10	9	8	**7**
s	16	16	15	14	14	15	15	14	13	12	11	9	9	**8**
t	17	17	16	15	15	15	15	15	14	13	12	10	10	**9**
s	18	18	17	16	16	16	16	16	15	14	13	10	11	**10**

Table 2.1: An example of a Levenshtein Edit Distance (LED) requiring 10 edits (with spaces removed)

Examples exist of using LED within MIR by Lewis et al. [2008] for the retrieval of scale and keys from a music database, for web content mining [zu Eissen and Stein, 2004], query-by-humming [Duda et al., 2007], annotation [Grachten et al., 2004] and for calculating melodic dissimilarity as part of MATT2 (Machine Annotation of Traditional Tunes) [Duggan et al., 2009]. A variation of LED where insertions and deletions are not allowed, called the Hamming Distance, is used in SEMEX, a music information retrieval system [Lemström and Perttu, 2000]. Edit distance has also been used for resolving spelling errors in lyrics [Müller et al., 2007]. An examination of LED distance functions for MIR can be found in [Lemström and Ukkonen, 2000].

2.2.2 Longest Common Substring

The Longest Common Substring (LCSStr) refers to the longest string that is a substring of both inputs and can be used as a measure of similarity. As such "verse" is a substring of "universe", "use" is not. The LCSStr is calculated differently from the LED in two ways, a) matching characters score 1 point and edits score 0 (the converse from the LED, hence the maximum score is sought), and b) only diagonal paths are considered. As with LED, a simple bottom-up algorithm for calculating the LCSStr in $O(N^2)$ time and space is shown in Algorithm 2 and an example of the cost matrix for two strings is shown in Table 2.2.

Input: String A, String B
Output: Length of Longest Common Substring $LCSStr$
Matrix m;
for $a \in [0..A.length]$ **do**
 | $m[a, 0] := (A[a] == B[0]? \ 1 : 0)$;
end
for $b \in [1..B.length]$ **do**
 | $m[0, b] := (B[b] == A[0]? \ 1 : 0)$;
end
for $a \in [1..A.length]$ **do**
 | **for** $b \in [1..B.length]$ **do**
 | | $m[a, b] := (A[a] == B[b]? \ 1 + m[a - 1, b - 1] : 0)$;
 | **end**
end
return $LCSStr := \max(m)$;

Algorithm 2: The Longest Common Substring.

	A	B	C	A	B
A	**1**	0	0	1	0
B	0	**2**	0	0	2
C	0	0	**3**	0	0
B	0	1	0	0	1
A	1	0	0	1	0

Table 2.2: An example of a Longest Common Substring (LCSStr) of 3.

LCSStrs have been used in MIR for indexing musical scores for musical queries [Medina, 2003] and in query-by-humming [Wu et al., 2006]. The DP algorithm above was chosen to show the similarities with other DP algorithms used in this text, however there are more efficient alternative DP implementations of LCSStr as well as other methods such as using a $O(N)$ suffix tree [Gusfield, 1997].

2.2.3 Longest Common Subsequence

The Longest Common Subsequence (LCSSeq) differs from the LCSStr in that subsequences can contain gaps (with no penalty for insertions or deletions). As such, both "verse" and "use" are subsequences of "universe". An example bottom-up algorithm for calculating the LCSSeq in Algorithm 3 uses a matrix that stores the lengths of subsequences, rather than calculating and storing all possible subsequences. An example of the matrix of lengths stored can be seen in Table 2.3.

Input: String A, String B
Output: Length of Longest Common Subsequence $LCSSeq$
$A := [0 + A]$;
$B := [0 + B]$;
Matrix $m := [0..A.length, 0..B.length]$;
for $a \in [0..A.length]$ **do**
 | $m[a, 0] := 0$;
end
for $b \in [0..B.length]$ **do**
 | $m[0, b] := 0$;
end
for $a \in [1..A.length]$ **do**
 | **for** $b \in [1..B.length]$ **do**
 | | $m[a, b] := (A[a] == B[b]?\ m[a-1, b-1] + 1 : \max(m[a-1, b], m[a, b-1]))$;
 | **end**
end
return $LCSSeq := m[A.length, B.length]$;

Algorithm 3: The Longest Common Subsequence.

LCSSeq is commonly used within MIR for matching tasks. Examples include LCSSeq being used for musical queries [Suyoto et al., 2007] and in the Fast Melody Finder [Rho and Hwang, 2006],

	0	A	B	C	A	B
0	**0**	0	0	0	0	0
A	0	**1**	1	1	1	1
B	0	1	**2**	2	2	2
C	0	1	2	**3**	3	3
B	0	1	2	**3**	3	4
A	0	1	2	3	**4**	**4**

Table 2.3: An example of a Longest Common Subsequence (LCSSeq) of 4.

and for measuring the similarity of lyrics as part of multimodal structure analysis [Cheng et al., 2009]. Lin et al. [2011] define a variation of LCSSeq, called rough longest common subsequence, for music matching. Additionally, a comparison of the efficiency of the most well known longest common subsequence algorithms can be found in Bergroth et al. [2000].

2.2.4 Dynamic Time Warping

Dynamic Time Warping (DTW) is used to synchronise two related streams of information bfy finding the lowest cost path linking feature sequences of the two streams together. This form of DP differs from the previously mentioned LED, LCSStr, and LCSSeq in that it allows any kind of sequences to be compared, so long as a suitable feature and cost metric is used to relate components of those sequences. Typically, when aligning music, an extra cost for insertion and deletion, *i.e.* one sequence waiting for another, is only used when finding the similarity is important, *i.e.* cover song detection, as this would punish repeats, skips, and differing tempo's whilst simultaneously reducing the chance of finding the optimal alignment. DTW has been widely used for audio synchronisation [Dixon, 2005], cover song identification [Serrà et al., 2008], automatic transcription [Turetsky and Ellis, 2003], speech processing [Sakoe and Chiba, 1978], gesture recognition [Müller, 2007], face recognition [Bhanu and Zhou, 2004], lip-reading [Murase and Sakai, 1996], data mining [Keogh and Pazzani, 2000], medicine [Vullings et al., 1998], analytical chemistry [Clifford et al., 2009], and genetics [Legrand et al., 2008], as well as in other areas.

DTW finds the minimal cost path through an accumulated cost matrix indexed by the elements of two sequences. In the case of audio, these sequences are typically audio features

from two input pieces. See Kirchhoff and Lerch [2011] for an evaluation of audio features for the alignment. From these two feature sequences $U = (u_1, u_2, ..., u_M)$ and $V = (v_1, v_2, ..., v_N)$, DTW finds the optimum path, $P = (p_1, p_2, ..., p_W)$, through the cost matrix $d_{U,V}(m, n)$, with $m \in [1, M]$ and $n \in [1, N]$, where each point $p_k = (m_k, n_k)$ indicates that frames u_{m_k} and v_{n_k} are part of the aligned path at position k. The final path is guaranteed to have the minimal overall cost, $D(P) = \sum_{k=1}^{W} d_{U,V}(m_k, n_k)$, whilst satisfying the boundary conditions $p_1 = (1, 1)$ and $p_W = (M, N)$, and the monotonicity conditions $m_{k+1} \geq m_k$ and $n_{k+1} \geq n_k$. The DTW algorithm, which has quadratic $O(N^2)$ computational costs, can be seen in Algorithm 4 and an example of DTW being used to align two chroma sequences can be seen in Figure 2.1.

Input: String A, String B
Output: Dynamic Time Warping DTW
Matrix m;
$m[0, 0] := cost(A[0], B[0])$;
for $a \in [1..A.length]$ **do**
  $m[a, 0] := cost(A[a], B[0]) + m[a - 1, 0]$;
end
for $b \in [1..B.length]$ **do**
  $m[0, b] := cost(A[0], B[b]) + m[0, b - 1]$;
end
for $a \in [1..A.length]$ **do**
  for $b \in [B..A.length]$ **do**
    $m[a, b] := cost(A[a], B[b]) + min(m[a - 1, b], m[a - 1, b - 1], m[a, b - 1])$;
  end
end
return $DTW := m[A.length, B.length]$;

Algorithm 4: Dynamic Time Warping.

Typically used to align music and music scores, [Soulez et al., 2003; Orio and Schwarz, 2001; Dannenberg, 1984], DTW is also used in many other areas of MIR such as audio matching [Dannenberg and Hu, 2003; Hu et al., 2003], singing voice segmentation [Xiao et al., 2008], lyrics and music alignment [Wong et al., 2007], structure analysis [Peeters, 2007; Chai, 2005], and query-by-singing [Tao et al., 2004]. DTW is used to align multiple MIDI files with music to established correspondences within the scores [Ewert et al., 2009].

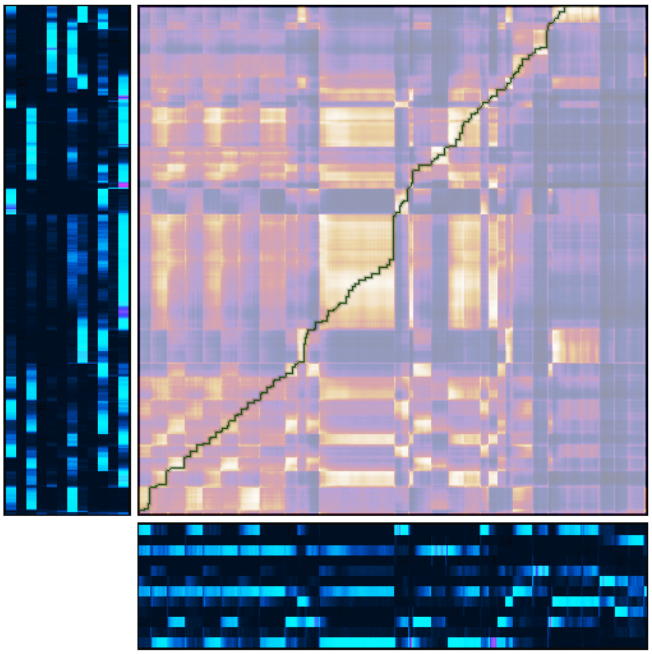


Figure 2.1: Dynamic Time Warping aligning two audio sequences. The audio is divided into chroma frames (bottom and left). The similarity matrix (centre) shows a path where the sequences have the lowest cost (highest similarity). Any point on this path indicates where in the progress of the music the corresponding audio relates to.

One drawback of DTW, that is common in many of the DP methods outlined here, is in its computational efficiency (see Section 2.2.7). There have been a number of attempts to improve DTW in this regard, including implementing a bounding limit on the path such as Sakoe and Chiba's bounds [Sakoe and Chiba, 1978] or Itakura's slope constraints [Itakura, 1975]. Salvador and Chan introduced FastDTW [Salvador and Chan, 2004] to use a multi-resolution DTW to bound a high resolution path within a set range of a low resolution path. FastDTW achieves near linear $O(N)$ computational and memory costs. Following on from this, Dixon showed how it is possible to build the accumulated cost matrix in an iterative and progressive manner and adapted and applied Sakoe and Chiba style bounds in Online Time Warping [Dixon, 2005] (OTW). This allowed DTW to not only be usable in real-time but also brought down computational costs to linear $O(N)$ costs, reducing the time taken to align larger

pieces. Dixon and Widmer [2005] then developed an audio-audio synchronisation application called MATCH that uses OTW and can synchronise two audio pieces in approximately 4% of the total time of the two pieces' durations with an average accuracy of 94.3% of notes within 200ms. MATCH was further extended by Arzt and Widmer [2010] using multiple instances of MATCH to process alignments at differing sections of the music so that the presented alignment could switch between them allowing omissions, forward and backward jumps, and unexpected repetitions.

2.2.5 Other Common DP Methods

Here we detail other DP methods commonly used in linking music and metadata related tasks. We do not make use of the following probabilistic methods as for each task they require an initial training process that makes them unsuitable for the general purpose music and metadata linking tasks we focus on.

Hidden Markov Models

Hidden Markov Models (HMMs) can be used to align music and metadata by modelling the audio by a sequence of observed states and the metadata as a hidden Markov chain. HMMs are dependent on a set of parameters, that can be set through training or by hand [Mauch, 2010], in order to control the transition probabilities. The Viterbi Algorithm [Viterbi, 1967] is used to find the most likely path through the hidden state space, given a sequence of observed states. HMMs have been typically used for real-time alignment tasks such as score following [Cano et al., 1999; Raphael, 2001; Schwarz et al., 2004; Cont and Schwarz, 2006] and speech recognition [Gales, 1998; Rabiner, 1990]. HMMs have also been used for recognising instruments [Kitahara et al., 2007], aligning lyrics [Wang et al., 2004; Mauch et al., 2010], and query-by-humming [Pardo and Birmingham, 2003]. Fang [2009] shows that HMMs and DTW are fundamentally based on the same DP principles and are interchangeable. Therefore, although we focus on developing and evaluating modifications for DTW over the next two Chapters, we expect this work to be applicable to HMMs as well.

Sequence Alignment in Bioinformatics

Within the field of genomics, DP is used to align protein or nucelotide sequences. Two prominent methods are the Needleman-Wunsch algorithm [Needleman and Wunsch, 1970] and SmithWaterman algorithm [Smith and Waterman, 1981] and build successively on the Levenshtein Edit Distance, although they are looking for the highest similarity score rather than the minimum cost. Based on bioinformatic principles of the similarity of certain amino acids/RNA/DNA combinations, the Needleman-Wunsch algorithm replaces the simplistic cost function of LED with a substitution matrix that favours certain combinations over others. Insertions and deletions are counted as gaps and penalized. The Smith-Waterman algorithm sets the negative scoring combinations in the similarity matrix to zero and once the matrix is calculated, the highest scoring cell is traced back to a zero cell to find the best sub-alignment.

2.2.6 Multiple Sequence Alignment

Multiple Sequence Alignment (MSA) [Corpet, 1988] is the use of DP methods to find co-occuring patterns across multiple sequences that is typically used in Biology to match protein, DNA and RNA sequences. In certain cases, MSA can make use of a hierarchical set of alignments of pairs of sequences. For instance, the LCSStr(A,B,C,D) can be calculated as LCSStr(LCSStr(A,B),LCSStr(C,D)). Knees et al. [2005] is an example of MSA being used to extract accurate lyrics from multiple sources.

2.2.7 Dynamic Programming Efficiency

The DP methods outlined in this section are all guaranteed to find the optimal solution to a specific problem by regarding all possible solutions. As mentioned, one common drawback to these methods is in their computational costs, both in processing and in memory.

Lazy evaluation, where processing is executed only when needed, has been shown to reduce the computational costs of LED [Tillström, 2010] and HMMs [Feldman et al., 2002]. Other means of reducing the time and space complexity of DP methods include using path constraints [Müller and Appelt, 2008; Sakoe and Chiba, 1978; Itakura, 1975] and multi-pass/multi-resolution [Salvador and Chan, 2004].

2.3 Research Trends

The following research map (Figure 2.2[20]), compiled by data mining Google Scholar,[21] provides a visualisation of the popularity of DP based methods within MIR. In total, 1160 pieces of research have been catalogued this way. HMM, GMM, HGM, DTW, LCSStr, LCSSeq, LED and other DP methods have been observed in MIR research related to music and metadata alignment. Papers and articles are shown as circles, colour coded depending on the method which is the focus of the paper, blue for HMM, green for GMM, *etc.* Additionally, the papers and articles have been arranged horizontally by date and vertically by the task which the paper/article is most relevant to. The size of the circle is related to the number of citations it has received, as a rough indicator of its influence. Additionally, citation links within the research shown are indicated by opaque grey lines linking the papers and articles.

From this map we can get an overview of how research in this field has been evolving over the last 16 years. Gaussian Mixture Models, for instance, are commonly used for **Classification** tasks with 44.1% of papers/articles in that category using GMMs. DTW is a big influence in **Score Following/Synchronisation**, responsible for 50.0% of the citations in those categories. The proportion of citations of papers/articles using HMMs drops from 32.8% in the 2000-2005 period to 15.5% in 2006-2010.

2.4 Applications of Linking Music Metadata

The following applications are made possible through developments in linking music metadata.

2.4.1 Score Tracking

A common use for synchronisation of metadata and digital music is that of online score tracking/following. By synchronising the live recording with the known musical score it becomes possible to follow a performer's position within a piece. The score is usually in Musical Instrument Digital Interface (MIDI) form but it can also be standard music notation, text, tablature

[20] An interactive version of this map is available at www.eecs.qmul.ac.uk/∼robertm/dpmir
[21] http://scholar.google.com/

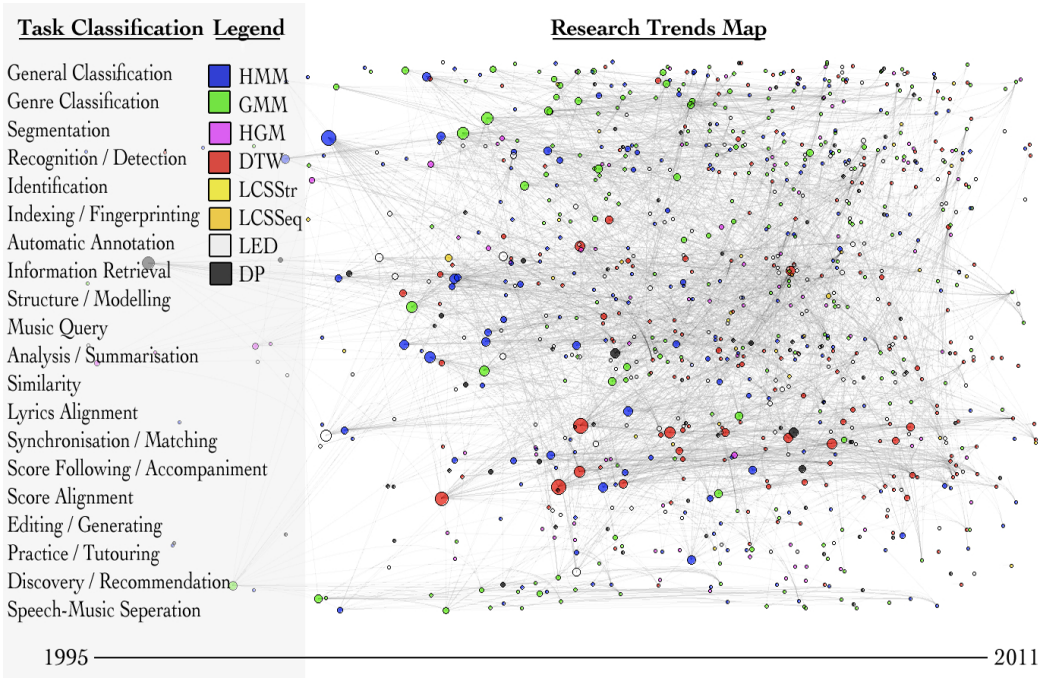


Figure 2.2: A map of research papers/articles corresponding to DP methods used in MIR. Circles relate to publications and are colour coded using the legend in the upper left region.

or other score format. The first example of score following forms part of Dannenberg [1984] automatic accompaniment system. Score tracking can lead to digital music books that turn pages for a musician automatically [Arzt et al., 2008].

The Probado System[22] [Berndt et al., 2010; Damm et al., 2011; Thomas et al., 2009], an extension of SyncPlayer [Kurth et al., 2005], is a digital library system that manages heterogeneous music collections (score, audio, lyrics) and incorporates MIR techniques such as score following and content based retrieval for document presentation by the Signal Processing Group at the university of Bonn led by Michael Clausen [Arifi et al., 2003, 2004; Kurth et al., 2007; Müller and Kurth, 2006; Müller et al., 2004] and the Bavarian State Library Probado Music Repository [Diet and Kurth, 2007]. Clausen et al. use this framework to demonstrate their methods which collectively represent a large part of the state-of-the-art research in score-

[22] http://www.probado.de/en_system.html

tracking and related applications.

MuseBook Score[23] is a commercial digital music book for pianists that displays MIDI files represented as sheet music and follows the musician's progress through the file, effectively turning the pages automatically. **MuseBook Score** can use either audio (through microphones) or MIDI input to track the user's performance.

2.4.2 Automatic Accompaniment

Automatic accompaniment is an extension to score following in order to provide accompaniment to a performance. This can be done using score following to drive the accompaniment or by directly synchronising pre-recorded audio with the musician. In this manner musicians can practice by themselves with the full sound of the overall performance or enhance live performances such as in Dannenbergs's On-Line Algorithm for Real-Time Accompaniment [Dannenberg, 1984]. **Music Plus One** by Raphael [2001] is an automatic accompaniment system that works in real-time using HMMs with pitch estimation for the observed features. The output is recorded or synthesised audio that plays in synchrony with a musician playing monophonic music.

2.4.3 Music Education

Music and metadata synchronisation has been used to create multimedia, interactive, musical education systems. Dannenberg et al. [1990] demonstrated the concept of a musical education system in the computer based **Piano Tutor** which inspired similar projects such as Software Toolworks' **Miracle Piano Teaching System**[24] and **PianoFORTE** by Smoliar et al. [1995].[25] These methods all make use of score-tracking to teach new piano students how to read music and play basic songs. The computer games industry has recently been producing music based games such as **Guitar Hero**, **Rock Band**, **Frets on Fire**,[26] or **Rock Prodigy**,[27] that attempt to make playing music more fun and accessible to gamers. Some of these have been very successful, selling millions of products. The technology used in these games has allowed

[23] http://www.musebook.com
[24] http://www.mobygames.com/game/miracle-piano-teaching-system
[25] http://www.informatik.umu.se/~jwworth/pianofor.html
[26] http://fretsonfire.sourceforge.net/
[27] http://www.rockprodigy.com

for novel interfaces for representing musical instructions but in doing so has greatly simplified the instruments and the music created to the point where the skills learnt are not transferrable to the actual instruments that they seek to recreate. **Piano Wizard**[28] by the Music Wizard Group has taken concepts from these games to provide a piano teaching application that evolves from a simplified game to actual score-reading as the users develop their knowledge.

2.4.4 Music Search using Music

Synchronising audio samples with metadata could allow for the quick and automatic collection of annotated data. Music could then be searched using music or annotations. **PeachNote**[29] by Viro [2011] is an internet search engine for standard music notation that allows users to search for specific sequences of notes [Viro, 2011]. **Shazam**[30] [Wang, 2006] and **SoundHound**[31] are smartphone applications for identifying songs from short audio clips of the music.

Audio that has been synchronised with musical scores can also be systematically gathered to provide ground truth data for testing other music information retrieval methods. The free availability of MIDI files of songs on the internet makes this a powerful method for gathering data. Turetsky and Ellis [2003] first used MIDI data aligned with audio to provide training data for automatic transcription methods. You and Dannenberg [2007] also used such data, gathered in a semi-supervised manner, as training data for note onset detection. As with MIDI, lyrics are also commonly available on the internet and can therefore be automatically gathered and synchronised with the matching music. This could then be used to train speech/vocal recognition methods in music. The linked lyrics can also be used as Karaoke data [Kan et al., 2008].

If a piece of music is aligned and linked to its musical score then that piece of music can be navigated by searching for features, segments [Viro, 2011] or lyrics [Kan et al., 2008]. The idea of the intelligent editor of digital audio goes back to Chafe et al. [1982] and Foster et al. [1982] who defined the need for music editing software that could allow users to interact with audio in high level musical terms. Synchronising the audio with the musical instructions, labels and other metadata would allow the music to be accessible through these linked features. Dannenberg

[28] http://www.pianowizard.com/
[29] http://www.peachnote.com/
[30] http://www.shazam.com/
[31] http://www.soundhound.com/

and Hu [2003] also describe an "Intelligent Audio Editor" that could automatically adjust audio segments to fit in with the overall mix as and when it receives them.

2.4.5 Beat/User Driven Music

Music or music videos can also be kept in synchronisation with the listeners' movements or beat. Allen and Dannenberg [1990] first implemented a real-time beat tracking prototype, followed by Masataka Goto [Goto and Muraoka, 1994] and [Goto and Muraoka, 1995]. **Cati Dance** [Jehan et al., 2003] proposes a beat-tracking approach that alters the tempo of a video clip of a woman dancing to match the tempo given by a beat-tracker for different audio clips. **B-Keeper** [Robertson and Plumbley, 2007] uses beat-tracking to keep a piece of music in sync with a drummer so that bands can use recorded audio segments that are automatically synchronised with a live performance.

BODiBEAT[32] is a personal audio player by Yamaha that is worn on a user's wrist and detects the running/jogging tempo of the user. It uses this tempo as a means to select which song stored on the device best fits the user's current tempo. If the user changes their tempo dramatically, the device will change track to find a new one that fits. An application for the iPhone called **SyncStep**[33] takes this concept further by allowing the user to select the music and adjusting the tempo of the music to fit that of a walker. When the user's tempo changes dramatically the music will change to the new tempo.

2.5 Conclusions

In this chapter we have reviewed research related to this thesis in the areas of metadata and Dynamic Programming. Additionally, we have shown how there are many potential uses, and opportunities, for mining and linking music audio and metadata. In sum, there is a great deal of music metadata being produced and made easily available on the web that is semantically relevant to the music itself and could enhance the listening or music learning experience. The Dynamic Programming methods reviewed here suffer from computational inefficiencies restricting their use within MIR but have the potential to make linking music and metadata and the

[32] http://www.yamaha.com/bodibeat/
[33] http://synchstep.com/

implicated applications possible. The challenges of the petabyte age require efficient metadata linking algorithms that can scale to large datasets. The rest of this thesis will be aimed at using and adapting the methods outlined in this chapter to link music and metadata in new and alternative ways. In the next chapter, we focus on modifications of Dynamic Programming to enable real-time synchronisation algorithms that can scale efficiently to align large datasets.

3

Real-time Dynamic Programming Methods for Music Alignment

"All musicians are subconsciously mathematicians." Thelonious Monk[1]

Contents

The process of aligning pairs of time series, such as music recordings with metadata, is a widely studied problem with applications in many fields. Automatic attempts at solving this task, in an off-line setting, commonly rely on DTW to find an optimal alignment between the time series. DTW suffers from three main drawbacks: that it is computationally inefficient, having time and space costs that are quadratic in the length of the series, that it is non-causal, which makes it unsuitable for alignment of signals in real-time or for large scale data mining, and that it assumes the sequences are complete matching sequences with similar starts and ends.

[1] http://jameslogancourier.org/index.php?itemid=5474

In this chapter we describe a number of modifications we apply to DTW in order to improve its efficiency and to make processing alignments in real-time possible.

In Section 2.2.4 we saw how previous improvements to DTW have enabled efficient processing but in a non-causal manner in FastDTW [Salvador and Chan, 2004] and a causal, real-time algorithm [Dixon, 2005]. Both algorithms restrict the length of the music they can process due to their memory requirements. FastDTW and On-Line Time Warping have linear memory costs which means that a piece of music that is sufficiently long will exhaust the memory capacity when using these methods to make alignments. Our modifications have linear memory costs for off-line alignment and constant memory costs for real-time synchronisation (discarding the path information) and are therefore not constrained in this manner.

Before we describe our adaptions of DP we first outline the different types of constraints we will make use of in Section 3.1. The first DP method is a non-recursive approach that we refer to as the Greedy method, and is described in Section 3.2. We then extend this method into the Two-Step algorithm in Section 3.3. We introduce Windowed Time Warping [Macrae and Dixon, 2010b], in Section 3.4. Modifications of On-Line Time Warping [Dixon, 2005] are described in Section 3.5, including On-Line Time Warping Jumping and in Section 3.6, On-Line Time Warping Constrained. We then examine the case for alignments not starting at the beginning of both sequences, as is commonly assumed to be the case, and describe an adaption to deal with these situations with our Quick Start Method in Section 3.7. We provide an example application of these methods in Section 3.8, before concluding this work in Section 3.9.

3.1 Constraints

There are various ways constraints can be applied in DTW. Constraints were proposed by Itakura [1975] and Sakoe and Chiba [1978] as a means of limiting the area of the similarity matrix through which a path may pass, in order to reduce the computational complexity of DTW. The simplest type of constraint is a global constraint on the path P, which is independent of the values in the similarity matrix. One example of such a constraint, illustrated in the rightmost graphic in Figure 3.2, restricts the maximum distance of the alignment path from the main diagonal of the similarity matrix [Sakoe and Chiba, 1978]. An extension of this idea is used in multi-resolution approaches such as FastDTW [Salvador and Chan, 2004], in which

a coarse-resolution DTW path is computed, and a fixed-width band around this path is then used to constrain the generation of successively higher resolution paths.

Alternatively, *local constraints* restrict the position of path points relative to neighbouring points, thereby determining the minimum and maximum slope of path segments and whether rows and/or columns of the similarity matrix may be skipped. Some examples of local constraints, as defined by Rabiner and Juang [1993], are shown in Figure 3.1.

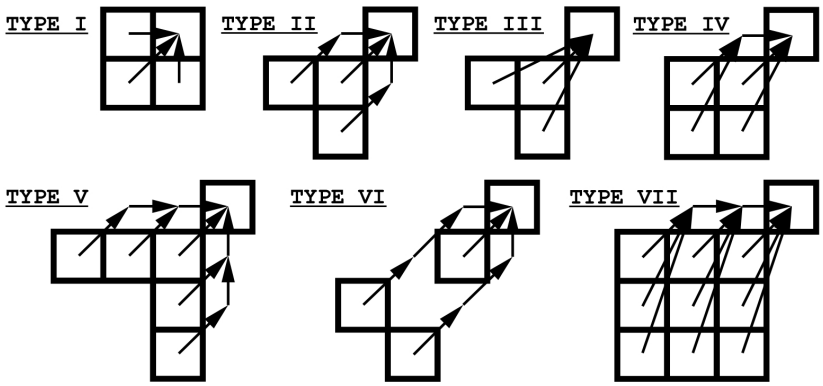


Figure 3.1: Examples of local constraints defined by Rabiner and Juang [1993].

These constraints may be expressed as a set of vectors $C = \{(m_{k+1} - m_k, n_{k+1} - n_k)\}$ indicating the possible steps from one path point $p_k = (m_k, n_k)$ to the next point $p_{k+1} = (m_{k+1}, n_{k+1})$. For example, the Type I local constraint in Figure 3.1 is expressed as $C = \{(1,0), (1,1), (0,1)\}$. In the following paragraphs, we distinguish two ways in which local constraints may be used.

3.1.1 Cost Constraint

A cost constraint, applied to a Dynamic Programming algorithm, defines which point within the similarity matrix a subsequent point originates from, based on the costs of the paths to the point which obey the local constraints. An accumulated cost matrix is computed from the similarity matrix such that every path point $p_k = (m_k, n_k)$ has a path cost $D(m_k, n_k)$ that is a combination of the difference cost $d_{U,V}(m_k, n_k)$ at that point and the minimum path cost to each of the possible preceding points $D(m_{k-1}, n_{k-1})$, where $(m_k - m_{k-1}, n_k - n_{k-1}) \in C$. For

example, a standard DTW using the Type I local constraint (from Figure 3.1), would have a path cost to any point (m_k, n_k) given by the following recursive definition:

$$D(m_k, n_k) = d_{U,V}(m_k, n_k) + \min_{(i,j)\in C} D(m_k - i, n_k - j), \tag{3.1}$$

where $D(1,1) = d_{U,V}(1,1)$.

3.1.2 Movement Constraint

A movement constraint defines possible successors to points on an alignment path, without reference to the context of overall path costs. For example, given a path point $p_k = (m_k, n_k)$, the subsequent path point p_{k+1} might be chosen on the basis of the similarity cost at that point:

$$p_{k+1} = (m_k, n_k) + \arg\min_{(i,j)\in C} d_{U,V}(m_k + i, n_k + j). \tag{3.2}$$

This defines a greedy (see Section 3.2) forward progression through the similarity matrix, which is useful to establish an upper bound on the minimum path cost. A movement constraint can also be used in conjunction with a cost constraint as a fast approximation to DTW, as described in 3.3 below. In Section 4.3 we evaluate a variety of constraints from [Rabiner and Juang, 1993], in addition to some of our own, for movement constraints, cost constraints, and combinations of both.

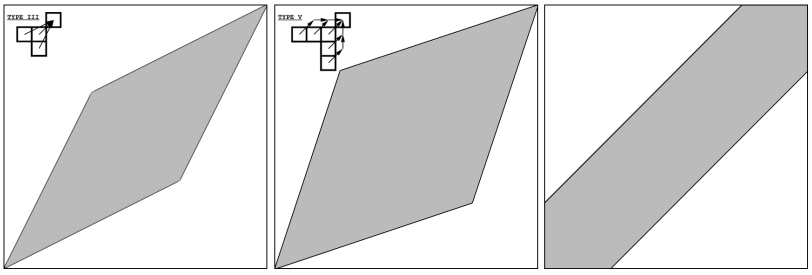


Figure 3.2: Potential path points within the Similarity Matrix for 3 types of constraint. From left to right: the Type III local constraint, the Type V local constraint and a global constraint.

3.2 Greedy

The simplest method for establishing a low-cost path is a greedy approach which extends a partial path by the lowest cost reachable point, without taking the accumulated cost into account (see Equation 3.2). The motivation for the Greedy method was to find the lowest possible computational cost for an alignment estimate. As such, the Greedy method uses only the most recent frames in the input sequences to calculate the subsequent alignment point.

A movement constraint (see Section 3.1.2) defines the possible range of relative points the Greedy path can move to. The Greedy method progressively calculates a path through the similarity matrix based on whichever subsequent point has the highest similarity (minimal cost) using a given movement constraint, describing the range of possible path points the path can move to. Each step is locally optimal, but it is unlikely to find the globally optimal path. However, it does provide an upper bound for other methods, and has a minimal computational cost. Latency in this Greedy method is minimal and dependent on the maximum number of possible movement frames in the movement constraint chosen. An illustration of the Greedy method can be seen in Figure 3.3.

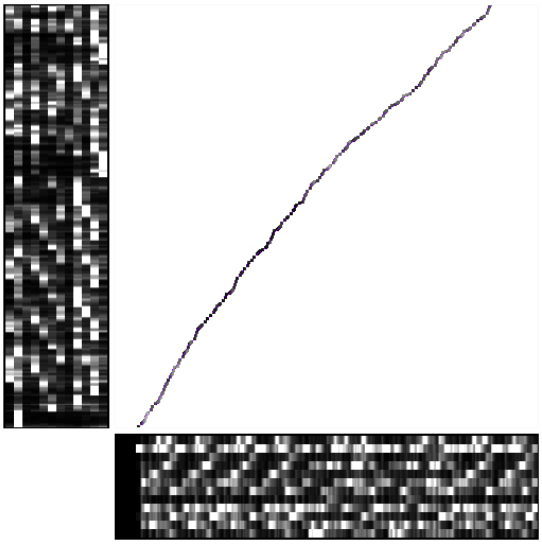


Figure 3.3: The Greedy path requires only potential points along the path within the scope of the movement constraint to be calculated.

3.3 Two-Step

The purpose of the Two-Step method is to allow a secondary adjustment to correct for errors in the alignment. This secondary step has been previously applied to OTW [Arzt et al., 2008]. The Two-Step method modifies the Greedy approach (Section 3.2) with the use of a second, cost constraint (see Algorithm 5). The extension of the path at each step is constrained by the movement constraint, but the cost of reaching each allowed point is determined by the cheapest known path leading to the point via one of the points determined by the cost constraint. The name of the method comes from the fact that the movement constraint determines a forward-looking set of path successor candidates, while the cost constraint provides the backward-looking search space for the cheapest path. The path determined in this way is monotonic (preserving the given order *i.e.* always positive), but as the calculation of cost can be based on alternative paths, the incremental path cost could be non-monotonic.

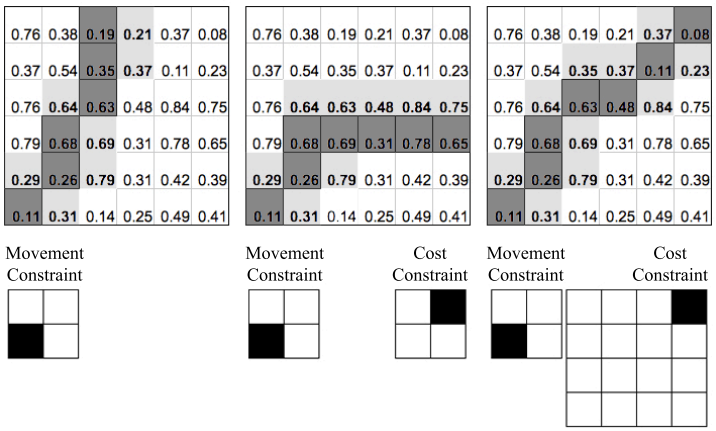


Figure 3.4: An example case of an alternative cost constraint helping a path avoid areas of noise (right) which would otherwise hinder a movement constraint only path (left) or an equal movement and cost constraint path (middle).

Two possible advantages of this method are envisaged: first, the non-monotonic path cost helps the path finding algorithm to recover from errors and not get trapped in local minima; second, the use of a low-latency (small maximum step size) movement constraint, necessary for real-time applications, can be combined with a cost constraint allowing larger steps, which might

offset the path accuracy problems caused by the limited context of the movement constraint. Figure 3.4 shows an example matrix which justifies the use of the two-step method with a wider cost constraint than the movement constraint. The Two-Step method was implemented using a linked list. Pseudo code for the Two-Step method can be seen in Algorithm 5.

Input: Feature Sequences A, B, Difference Matrix $d_{A,B}$, Cost Constraint C_c, Movement Constraint C_m
Output: Path $P = \{p_1, p_2, ...\}$ where $p_k \equiv (m_k, n_k)$
$k := 1$;
$p_k := (1, 1)$;
$D(p_k) := d(p_k)$;
while $m_k < A.length$ ***and*** $n_k < B.length$ **do**
  Best := Null;
  for $(i, j) \in C_m$ **do**
    Test := $(m_k + i, n_k + j)$;
    if *D(Test) = Null* **then**
      for $(u, v) \in C_c$ **do**
        if $D(m_k + i - u, n_k + j - v) \neq Null$ **then**
          Cost := $D(m_k + i - u, n_k + j - v) + d_{A,B}(m_k + i, n_k + j)$;
          if *D(Test)=Null* ***or*** *Cost<D(Test)* **then**
            D(Test) := Cost;
          end
        end
      end
    end
    if *Best = Null* ***or*** *D(Test) < D(Best)* **then**
      Best := Test;
    end
  end
  $k := k + 1$;
  p_k := Best;
end
return $P := \{p_1, p_2, ..., p_k\}$;

Algorithm 5: The Two-Step Method.

3.4 Windowed Time Warping

Windowed Time Warping (WTW) [Macrae and Dixon, 2010b] is a linear cost variation on DTW for real-time synchronisation. WTW consists of calculating small sub-alignments and combining these to form an overall path. Subsequent sub-paths are started from points along the previous sub-paths. The end points of these sub-alignments are either undirected, by assuming they lie on a given diagonal, or directed, by using a forward path estimate. In practice, undirected means selecting a set distance ahead of the current position in both sequences as the end point for the subsequent window, whereas directed would make use of the Greedy or Two-Step method to calculate where such end points are likely to be. As such WTW can be seen as a two-pass system similar to FastDTW [Salvador and Chan, 2004] and OTW [Dixon, 2005]. The motivation for WTW is based on the principles of Dynamic Programming in dividing large problems into smaller problems. Additionally, the sub-alignments make use of an optimisation that avoids calculating points with costs that are over the cost estimate (provided by the initial direction path), referred to as the A-Star Cost Matrix (Section 3.4.4). The overall process is outlined in Algorithm 6.

3.4.1 Windowing the Alignment

The Windowed Time Warping (WTW) method divides the alignment into a series of blocks of frames ("windows") that are aligned in order, using DTW, as the audio data is received. In a similar manner to how audio data is segmented into overlapping frames, the windows in WTW have a window size and hop size to describe their size and spacing respectively, with a segment of the global alignment being computed via DTW for each window. A larger window size and/or smaller hop size will increase the accuracy of the alignment, as more of the cost matrix is calculated, however this will be less efficient. Examples of different window and hop sizes can be seen in Figure 3.5. The sequence of windows that make up the alignment of WTW can be either directed or undirected. To direct WTW the Greedy method mentioned above can be used to guide the small-scale standard DTW alignments.

Input: Feature Sequences A, B, Difference Matrix $d_{A,B}$, Constraint C, Window Size w, Hop Size h
Output: Path $P = \{p_1, p_2, ...\}$ where $p_k \equiv (m_k, n_k)$
$k := 1$;
$p_k := (1, 1)$;
while $m_k < A.length$ ***and*** $n_k < B.length$ **do**
  $q_1 := p_k$;
  for $i := 2$ ***to*** w **do**
    $q_i := \arg\min_{c \in C}\ d_{A,B}(q_{i-1} + c)$;
  end
  $\{r_j\}_{j=1,2,...} := \mathrm{DTW}(q_1, q_w, C)$;
  for $i := 1$ ***to*** h **do**
    $p_{k+i} := r_i$;
  end
  $k := k + h$;
end
return P;

Algorithm 6: The basic Windowed Time Warping algorithm, directed by the Greedy algorithm. $\mathrm{DTW}(a, b, C)$ is the Dynamic Time Warping algorithm, with start point a, end point b and cost constraint C.

3.4.2 Window Guidance

The sequential windows that make up the alignment of WTW can be either directed or undirected. Whilst it can help to direct the end point of the windows of DTW (particularly for alignments between sequences where the expected path angle will be far from 45°), the sub-paths calculated within these windows can make up for an error in the estimation. A low hop size should ensure the point taken from the sub-path as the starting point for the next window is likely to be on the correct path.

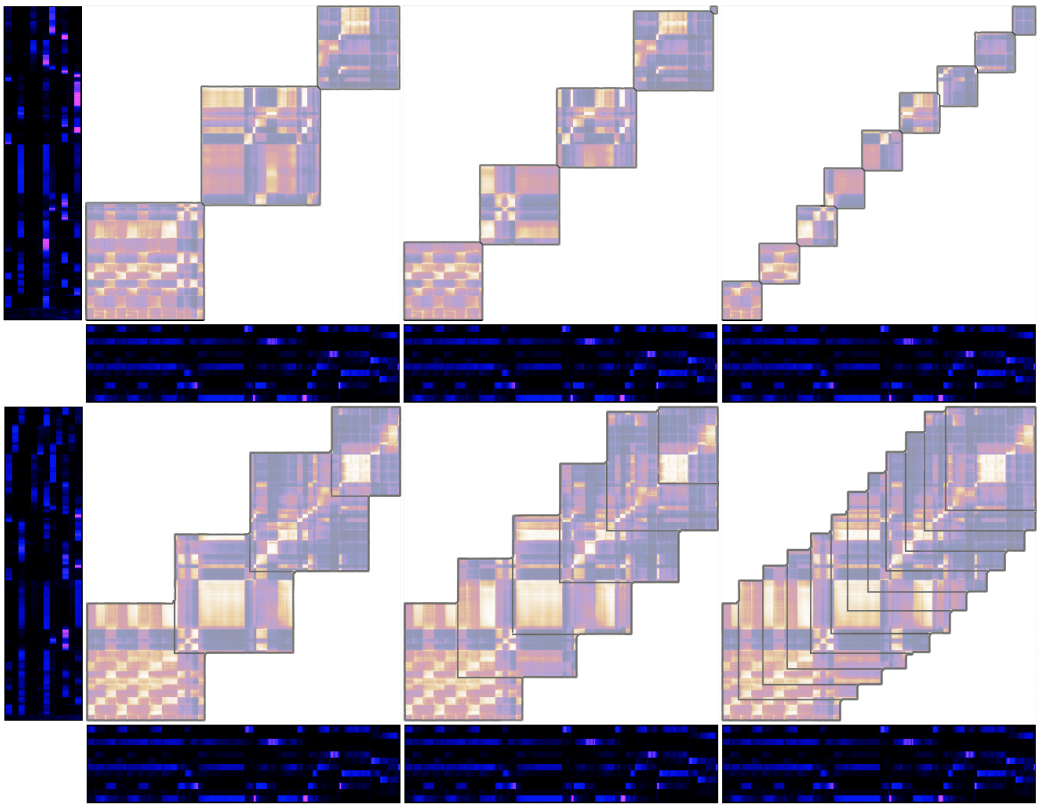


Figure 3.5: The regions of the similarity matrix computed for various values of the window size (top row) and hop size (bottom row).

For the windows to be directed, a forward estimation is required. The Greedy method (Section 3.2) can be used to calculate an efficient forward path estimate in a progressive manner. The first forward path estimate $F = (f_1, f_2, ..., f_W)$ where $f_k = (m_k, n_k)$ starts from position $f_1 = (m_1, n_1)$.

The forward path estimate only needs to calculate similarities between frames considered within the local constraints and so at this stage a vast majority of the similarity matrix does not need to be calculated. When the forward path estimate reaches f_W, the WTW window size, the final point $f_W = (m_W, n_W)$ is selected as the end point for the accumulated cost-matrix.

Note that some combinations of constraints that skip points (*e.g.* where i or j are greater

than 1) will require that jumps in the forward path are filled in order to compute a complete path cost estimate, as in the Type V local constraint, so that the cost estimation F is complete.

3.4.3 Accumulated Cost Matrix

The windowed area selected is calculated as an accumulated cost matrix between the beginning and end points of the forward path estimate *i.e.* $\mathcal{C}(m,n)$ of $m \in [m_{f_1} : m_{f_W}]$ and $n \in [n_{f_1} : n_{f_W}]$. This accumulated cost matrix can be calculated in either a forward or reverse manner, linking the start to the end point or vice versa. This uses the standard Type I cost constraint to determine a frame's accumulated cost, given by:

$$D(m,n) = d(m,n) + \min \begin{Bmatrix} D(m+1,n+1) \\ D(m+1,n) \\ D(m,n+1) \end{Bmatrix} \tag{3.3}$$

where each sub-path is treated as a separate path with an initial cost of $D(S_1) = 0$. The sub-path $Si = (s_{i,1}, s_{i,2}, ..., s_{i,H})$ is given by the prefix of the minimum cost path of length V, the WTW hop size, H. The final point, $s_{i,H}$, is taken as the starting point for the next window. The sub-paths are concatenated to construct the global WTW path *i.e.* $P = (S_1, S_2, ..., S_L)$. The final sub-path S_L occurs when the forward path estimate reaches the ends of both input sequences, u_M and u_N and this sub-path is not constrained by the WTW hop size.

3.4.4 A-Star Cost Matrix

Either of the undirected and directed window end point estimations provide an estimate cost $D(f_H)$ for each sub-path S_i. This estimate can be used to disregard any paths within the accumulated cost matrix that are above this cost as it is known that there is a path that is cheaper. The calculation of the similarity for many of these paths can be avoided by calculating the accumulated cost matrix in rows and columns from the end point f_L to the start f_1. When each possible preceding point for the next step of the current row/column has a total cost above the estimated cost, *e.g.* $\min(D(m+1,n+1), D(m+1,n), D(m,n+1)) >= D(G)$, the rest of the row/column is then set as more than the cost estimate, thus avoiding calculating the

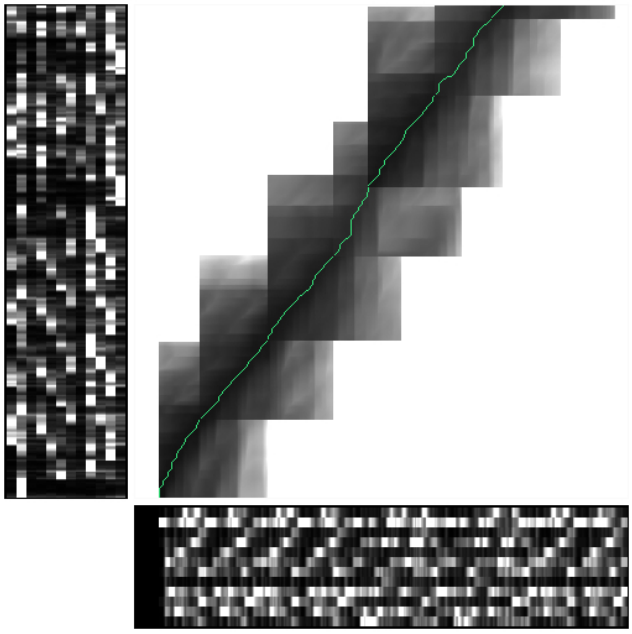
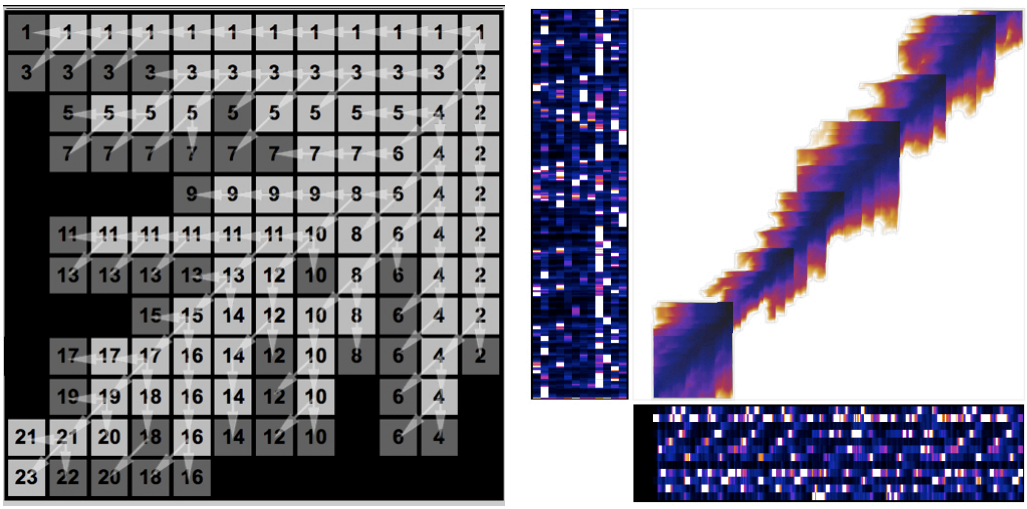


Figure 3.6: The Windowed Time Warping path.

accumulated cost for a portion of the matrix. This procedure can be seen in Figure 3.7 (a). An example of the final WTW path is shown in Figure 3.7 (b).

3.5 On-Line Time Warping Jumping

OTW-Jumping is a score following modification of MATCH [Dixon and Widmer, 2005] and is based on OTW [Dixon, 2005]. Partial rows and columns of the accumulated cost matrix are calculated incrementally as required, with the choice of row or column being determined by the end point of the minimum-cost path to any point in the most recently computed row or column (see Figure 3.8 (a)). As an example, a new column of accumulated cost points will be calculated when the lowest cost path ends in the middle of the last-computed column (step 19 in Figure 3.8 (a)). There is also a limit for the length of these rows and columns, which defines the width of the band in which the path exists, similar to a dynamic version of the Sakoe and Chiba constraint [Sakoe and Chiba, 1978]. This also determines the largest possible jump the

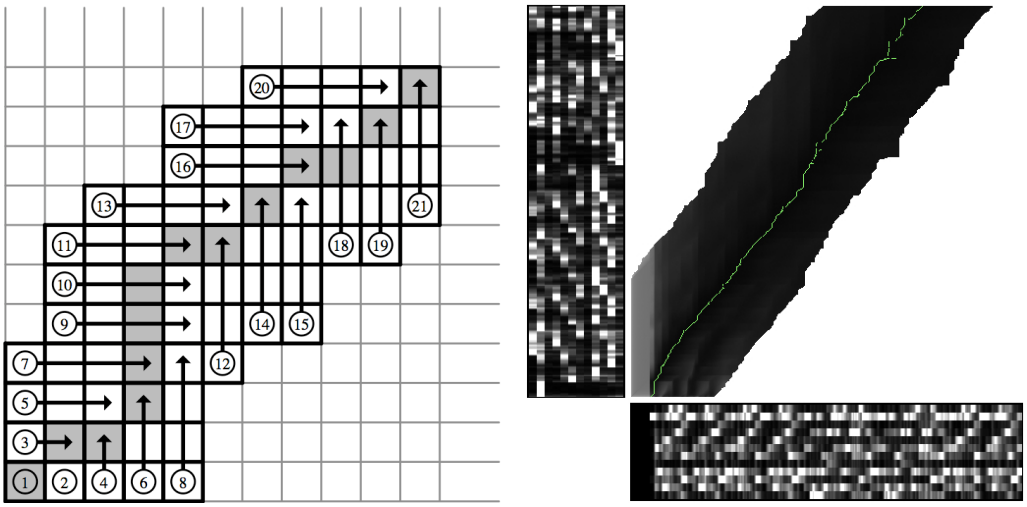


(a) (b)

Figure 3.7: (a) The calculation of the accumulated cost matrix. The numbering shows the order in which rows and columns are calculated and the progression of the path finding algorithm is shown by arrows. Dark squares represent a total cost greater than the estimated path cost whilst black squares indicate points in the accumulated cost matrix that do not need to be calculated. (b) The area of the cost matrix calculated when the A-Star Cost Matrix modification is applied to Windowed Time Warping.

synchronisation can make.

The modifications to the original OTW method include changing the bias so that horizontal, vertical and diagonal steps are considered equally and allowing the path to move to any point on the most recently expanded row or column within the known path points. Therefore the path can 'jump' and does not satisfy the typical monotonicity condition $m_{k+1} \geq m_k$ and $n_{k+1} \geq n_k$ for all $k \in [1, W-1]$. As it is not necessary for all the possible paths to be kept in memory, OTW-Jumping benefits from being less computationally expensive than the original MATCH method. The OTW-Jumping method is shown in Algorithm 7 and an example of the final path can be seen in Figure 3.8 (b).



(a) (b)

Figure 3.8: (a) An example showing the order of computation of partial rows and columns of the accumulated cost matrix using on-line time warping [Dixon, 2005]. (b) The modified On-Line Time Warping Jumping path.

3.6 On-Line Time Warping Constrained

OTW-Constrained is a variation on OTW which avoids latency and retains monotonicity by using a single step movement constraint similar to the Type I local constraint in Figure 3.1. However, unlike the typical local constraint that selects the point with the lowest accumulated cost, the OTW-Constrained method extends an incrementally computed path by selecting, from the points reachable via the local path constraint, the point which is closest to the point with lowest accumulated cost. An example of the OTW-Constrained path can be seen in Figure 3.9.

3.7 The Quick Start Method

It is usually assumed that music and metadata sequences that need to be aligned relate to each other as full sequences. However, in practice, one sequence might relate to a sub sequence of the other and therefore the alignment between pieces would not necessarily start at the beginning of both pieces. The Quick Start Method [Macrae and Dixon, 2010b] is an attempt to

Input: Feature Sequences A, B, Accumulated Difference Matrix $D_{A,B}$, Band Width b
Output: Path $P = \{p_1, p_2, ...\}$ where $p_k \equiv (m_k, n_k)$
$k := 1$;
$p_k := (1, 1)$;
$(r, c) := p_k$;
while $m_k < A.length$ ***and*** $n_k < B.length$ **do**
  if $m_k = r$ **then**
    $r := r + 1$;
    for $v := c - b + 1$ ***to*** c **do**
      Compute $D_{A,B}(r, v)$;
    end
  end
  if $n_k = c$ **then**
    $c := c + 1$;
    for $u := r - b + 1$ ***to*** r **do**
      Compute $D_{A,B}(u, c)$;
    end
  end
  $k := k + 1$;
  $p_k = \arg\min_{u=r \text{ or } v=c} D_{A,B}(u, v)$;
end
return P;

Algorithm 7: On-Line Time Warping Jumping. The accumulated costs $D_{A,B}(u, v)$ are computed according to Equation 3.1, with the recursion restricted to points which have a previously computed accumulated cost.

locate the best starting trajectory for an alignment that considers any possible starting point in sequences $U = (u_1, u_2, ..., u_M)$ and $V = (v_1, v_2, ..., v_N)$. The Quick Start Method uses multiple occurrences of the Greedy method (Section 3.2) starting at every possible position along each sequence with the other at the initial frame *e.g.* (u_1, u_n) or (u_m, v_1). Then, as the multiple Greedy paths take each simultaneous step forward, all the paths are pruned so that all paths for which the overall cost $D(P_i)$ is above the average cost are discarded. When two paths collide, the path with the higher cost is discarded. This selective process is suspended during silent frames otherwise paths would be pruned without any meaningful cost data to base the path pruning on. The last remaining path is considered to be the correct initial alignment. Figure

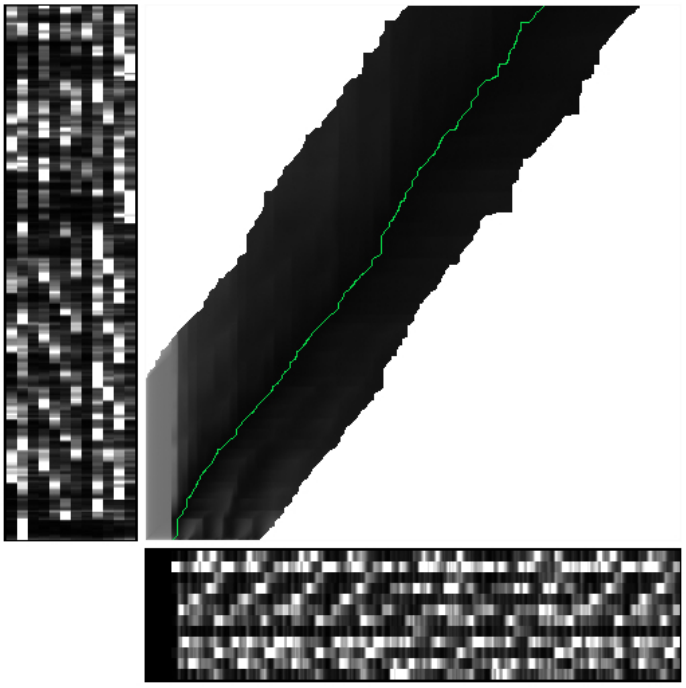


Figure 3.9: The On-Line Time Warping Constrained path. In this instance the area of the similarity matrix calculated is similar to the On-Line Time Warping Jumping path in Figure 3.8 (b), however the path itself is continuous as it is constrained to single steps.

3.10 shows the path pruning effect on multiple paths.

3.8 MetaSync

MetaSync[2] is a Java application with a Max/MSP[3] interface that implements each of the alignment methods described in this chapter including traditional DTW. MetaSync can align two audio streams or one audio stream with a MIDI score file. The user can select which feature to use from Chroma, MFCCs and Tonal Centroids [Harte et al., 2006] and the window and hop size dimensions. The output can be graphical, synthesised MIDI or a text list. For example, Figures 2.1, 3.3, 3.5, 3.6, 3.7 (b), 3.8 (b), 3.9, and 3.10 were produced using MetaSync.

[2]http://robmacrae.com/metasync
[3]http://cycling74.com/products/maxmspjitter/

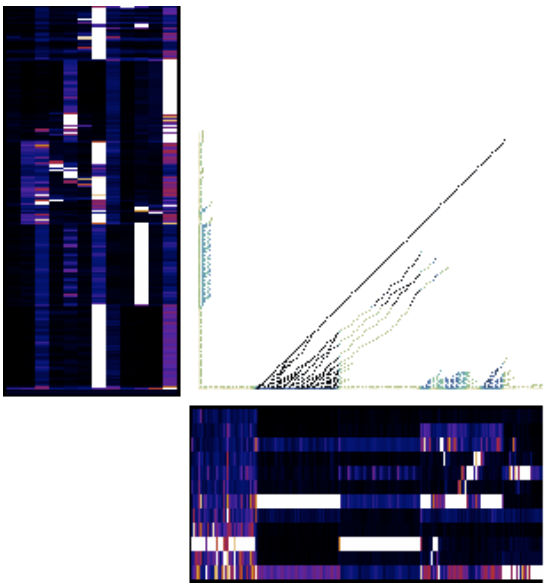


Figure 3.10: The Quick Start Method.

3.9 Conclusions

Dynamic Programming methods are typically only used for off-line alignment tasks on small datasets. Quadratic computational costs and a requirement of knowing the full input sequences hinder DP methods from running in real-time or scaling efficiently over big data. In this chapter we have presented five new[4] linear time modifications of DP that can run in real-time with variable levels of latency. Additionally, we have proposed an alternative starting mechanism incorporating DP based audio segment match to start synchronisations at the right position within the input sequences. The Greedy and Two-Step methods, and WTW, each utilise a movement constraint and/or cost constraint. These constraints can be adapted to affect the amount of the cost matrix considered in forward paths and which points are considered when calculating the path dynamically. WTW, OTW-Jumping and OTW-Constrained each have path "corridors", the size of which is controlled by a bounding limit. This bounding limit affects the amount of the cost matrix considered in the alignment and the ability of the paths

[4]It is likely that the Greedy method has been used previously however we were unable to find a named algorithm matching its description.

to handle jumps in the input sequences. In the next chapter we will evaluate the alignment methods within the context of a score alignment system and explore the trade off between latency and accuracy with various configurations of the cost constraints and path bounding limits.

4
Evaluation

"Music is the pleasure the human mind experiences from counting without being aware that it is counting." Gottfried Leibniz[1]

Contents

In this chapter we focus on the evaluation of DP methods for the music and metadata linking task of score following, or its off-line equivalent, score alignment. Evaluation of score following/alignment methods is problematic as manually aligned test data is required but difficult to author. It is possible to use MIDI synthesis to generate audio test data from an altered copy of the MIDI with the purpose being to rediscover the alteration using the original MIDI as the score[Kirchhoff and Lerch, 2011], however this method does not give a realistic estimate of performance on natural recordings, as synthetic data is easier to align than audio recordings. Previous evaluations of score following techniques typically used hand-annotated data containing small quantities of test pieces. In this chapter, we examine how score following test data is authored and develop a new tool to assist the creation of score following test data.

[1] http://www.bbc.co.uk/programmes/p003c1b9

Using this tool, we adapt two datasets to create automatic and manually annotated datasets in Section 4.1. In Section 4.2 we describe the evaluation metrics used to judge the accuracy of the methods. We compare different cost and movement constraints for the Two-Step Method in Section 4.3. We then evaluate the score following methods from Chapter 3 in Section 4.4. In order to test the computational costs of these methods, we test using various length sequences and make comparisons with FastDTW in Section 4.5. We compare the manually produced and the automatic test data in Section 4.6 before drawing our conclusions in Section 4.7.

4.1 Authoring Test Data

Ground truth data for music and score synchronisation/alignment methods consists of a collection of audio files in either compressed (*e.g.* MP3) or uncompressed (*e.g.* WAV and PCM) formats, and their corresponding musical scores (in MIDI format), as well as reference files (plain text) that specify the correspondences between notes (or beats) in the audio and score representations. It is possible for the reference file to also be in MIDI format and represent a synchronised score of the audio.

There are three commonly used methods to appropriate test data for score following purposes. The first is by marking the data manually, which requires considerable human effort [Cont et al., 2007]. The second is by synthesising audio from the score and modifying either the audio or score timing and testing the algorithms' ability to find the modifications via alignment [Turetsky and Ellis, 2003; Kirchhoff and Lerch, 2011]. The third is to use data produced by off-line alignment as a substitute for ground truth [Macrae et al., 2010]. The second and third approaches could positively bias the results, as synthesised audio is cleaner and easier to process than natural recordings, and off-line alignment uses similar features and methods as the algorithms being evaluated.

4.1.1 MetaSync and Sonic Visualiser

In order to assist with authoring score following test data we extended MetaSync to produce session files for Sonic Visualiser.[2] Sonic Visualiser is a music analysis program developed at C4DM (The Centre for Digital Music). This output of MetaSync allows users to create session

[2]Sonic Visualiser is available at http://www.sonicvisualiser.org

files from a reference dataset to analyse the reference, MIDI and music audio simultaneously. These session files, when imported into Sonic Visualiser, show an altered version of the score (modified to fit the alignment points, if given), plotted on top of a chroma view of the audio. This visual display allows users to examine any discrepancies in the reference data. An example of this output being used in Sonic Visualiser can be seen in Figure 4.1.

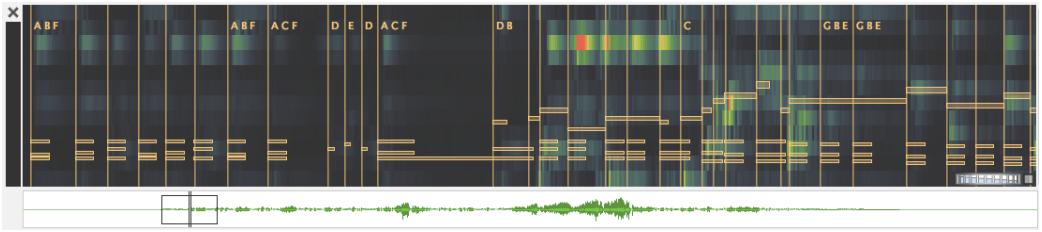


Figure 4.1: Using the Alignment Visualiser tool in Sonic Visualiser to check reference onset times. On one layer is a 12 dimensional chroma view of the audio sequence data with a hop size of 20 ms. Overlaid on this is a piano-roll view of the score, with timing altered according to the reference alignment. The lines mark the onset times of the score notes and should correspond to a matching onset time in the audio.

4.1.2 Test Data

In order to evaluate the accuracy of the proposed methods, we require a ground truth set of alignment points with which to compare the alignment paths produced by the selected methods. These alignments consist of note onset times for each score note, *i.e.* alignment points that should lie on the path identified by each algorithm. For each reference point, we judge whether the correct alignment was found, relative to five levels of accuracy that specify the maximum difference between the known onset time and the corresponding coordinate on the alignment path. This shows the accuracy of the methods at various resolutions, which informs the choice of alignment methods for applications which may have specific requirements. For the purposes of this evaluation, the five accuracy levels are 25, 100, 200, 500 and 2000 ms.

This evaluation used two datasets that were each annotated twice, once involving a large degree of manual marking of the note onset times and once by automatically aligning the data using a standard off-line method. This allows us to compare three sets of evaluation points

for each dataset: the two produced by the above-mentioned approaches alone, and a third set containing the subset of data points where both methods agree. These datasets come from the 2006 MIREX Score Following evaluation [Downie et al., 2005] and the CHARM (Centre for the History and Analysis of Recorded Music) Mazurka project [Cook, 2007].

The Mazurka Score Following Dataset

The CHARM Mazurka Project collected over 2,500 performances of 49 of Chopin's Mazurkas (solo piano music) in order to analyse performance interpretation. For a portion of these recordings, "reverse conducting" data has been made available,[3] consisting of beat times annotated by Craig Sapp [Sapp, 2007] using a tool which records the user tapping the beats while listening to the audio. To improve reliability, the process was repeated up to 20 times for each recording, and the average tap time for each beat was taken. A score to score alignment was necessary to link the events in the reverse conducting data to those in the score. A combined set of 355 audio, midi and reference files were collected.

MIREX 06 data

The first public evaluation of score following systems took place at the 2006 Music Information Retrieval Evaluation Exchange (MIREX) and included a dataset of 46 short monophonic pieces aligned by Arshia Cont [Cont et al., 2007] that were initially automatically aligned with an off-line standard DTW and then corrected by hand with the help of an onset detector. This MIREX database consists of 46 audio files consisting of flute, violin, clarinet and singing, with their corresponding score and reference files.

A summary of the two datasets and some of their basic attributes are shown in Table 4.1.

Dataset	No. of pieces	Average length (s)	Polyphonic	Genre	Instrument
MIREX 06	46	63.2	No	Varied	Varied
MAZURKA	355	152.1	Yes	Classical	Piano

Table 4.1: Summary of the test data used in the evaluation.

[3] http://mazurka.org.uk/info/revcond/

A secondary automatic alignment is based on Dan Ellis's DTW Matlab Toolbox[4] [Turetsky and Ellis, 2003], which performs alignment using chroma features. The chroma features were extracted with the Toolbox using a window size of 80 ms and hop size of 20 ms. For the third, combined set, we select the manual onset times that are within the accuracy required for each accuracy level used in the testing. Table 4.2 shows the percentage of onset times for which the two methods agree. The closer agreement of the MIREX 06 data can be explained by the use of a similar DTW technique as the basis for the manual annotation.

	Accuracy level (ms)				
Dataset	25	100	200	500	2000
MIREX 06	53.7%	85.9%	90.5%	95.3%	98.0%
MAZURKA	46.4%	75.7%	86.1%	95.2%	99.6%

Table 4.2: Agreement rates between manual and automatically produced alignments at various accuracy levels.

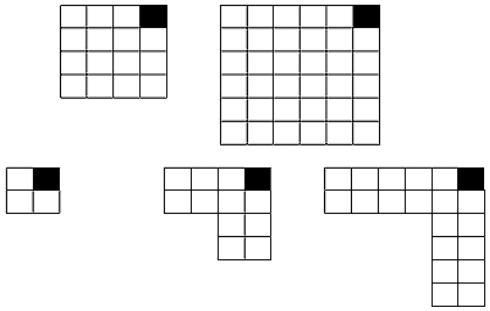
4.2 Evaluation Metrics

When evaluating alignment methods we look how closely the points on the alignment path resemble the known alignment points in the ground truth data. The alignment points were interpolated and averaged as required so that for any given time in the score there was a corresponding time in the audio data. In this manner, frames that do not contain an event are ignored, reflecting the ambiguous nature of synchronising data between musical onsets. Finally, the difference between the computed audio time and the reference audio time is compared against 5 different accuracy levels: 25, 100, 200, 500 and 2000 ms. The proportion of events with an error less than or equal to each accuracy level gives an accuracy rating for each piece and level. These piece-wise accuracies are then averaged to give overall accuracies for each dataset, algorithm and accuracy level.

[4]The http://labrosa.ee.columbia.edu/matlab/dtw/

4.3 Two-Step Constraint Calibration

A comparison of various local constraints used as the movement constraint and the cost constraint for the Two-Step method is shown in Figure 4.2.



	Local constraint combinations: Accuracies as percentage of notes located accurately to within 2000 ms				
	(A)	(B)	(C)	(D)	(E)
(A)	35.5	30.8	30.7	28.4	28.9
(B)	36.8	35.5	33.7	36.6	34.4
(C)	39.2	36.7	35.4	40.4	38.5
(D)	36.8	35.6	33.8	36.5	34.5
(E)	39.2	36.7	35.4	39.4	38.8

Figure 4.2: This table and the graphics above it show the accuracy results for the Two-Step path finding method for various combinations of local constraints. The graphics show the constraints used with the white boxes representing the points in the similarity matrix from which the black point can be reached. The black points are aligned with their respective columns in the table. The rows in the table represent the different constraints for the path's movement constraint and the columns represent the path's cost constraint. The results are from both datasets combined at the 2000 ms accuracy level.

The results indicate which attributes of the local constraints affect alignment. For example, the constraints that allow vertical and horizontal movement performed best with our datasets. This is related to the fact that the music has highly varying tempo, including pauses, which corresponds to path segments of extreme (high or low) gradient. In general, the larger the steps allowed by the movement and cost constraints, the better the results. The disadvantage is that increasing the maximum step size of the movement constraint also increases the latency

of the algorithm. The results show that the Two-Step method with a low latency movement constraint can benefit from a less restrictive cost constraint to improve accuracy as in the case of movement constraint C and cost constraint D.

4.4 Path Strategy Evaluation

In this section we evaluate the five score following methods from Chapter 3 as well as two variations of MATCH [Dixon and Widmer, 2005].

On-Line Time Warping MATCH (Forward Path)

The original OTW application MATCH is included here for comparison purposes. For MATCH the MIDI files are synthesised first as MATCH aligns audio files. Unlike the other methods, MATCH has fixed local slope limits and uses an alternative feature based on mapping the spectrum into 84 dimensions with the low end linearly scaled and the high end logarithmically scaled. We use two different configurations of MATCH. The first is the causal algorithm which uses the forward (zero-latency) path (Match-Forward), corresponding to the points (r, c) computed in each iteration of the main loop of Algorithm 7.

On-Line Time Warping MATCH (Backward Path)

The second configuration of MATCH is the non-causal (off-line) algorithm which, like DTW, estimates the optimal path backwards after all data has been processed (Match-Back).

The results of the evaluation of the various path strategies are compared using three alternative reference datasets (Manual, DTW, and Combined) in Figure 4.3. The WTW, OTW-Jumping and OTW-Constrained methods' bounding limits are set at 10 seconds. For the Greedy, Two-Step and Windowed tests, movement constraint C and cost constraint D were used. From these diagrams we make several observations. With regard to the path strategies, there are contrasting results depending on the dataset used, whereby the WTW method performs comparatively strongly on the MIREX 06 data yet less so with the MAZURKA data.
Of the real-time methods, WTW offers the best accuracy rates at the higher accuracy level of 25 ms but not at the lowest accuracy requirement of 2000 ms. The methods which use OTW

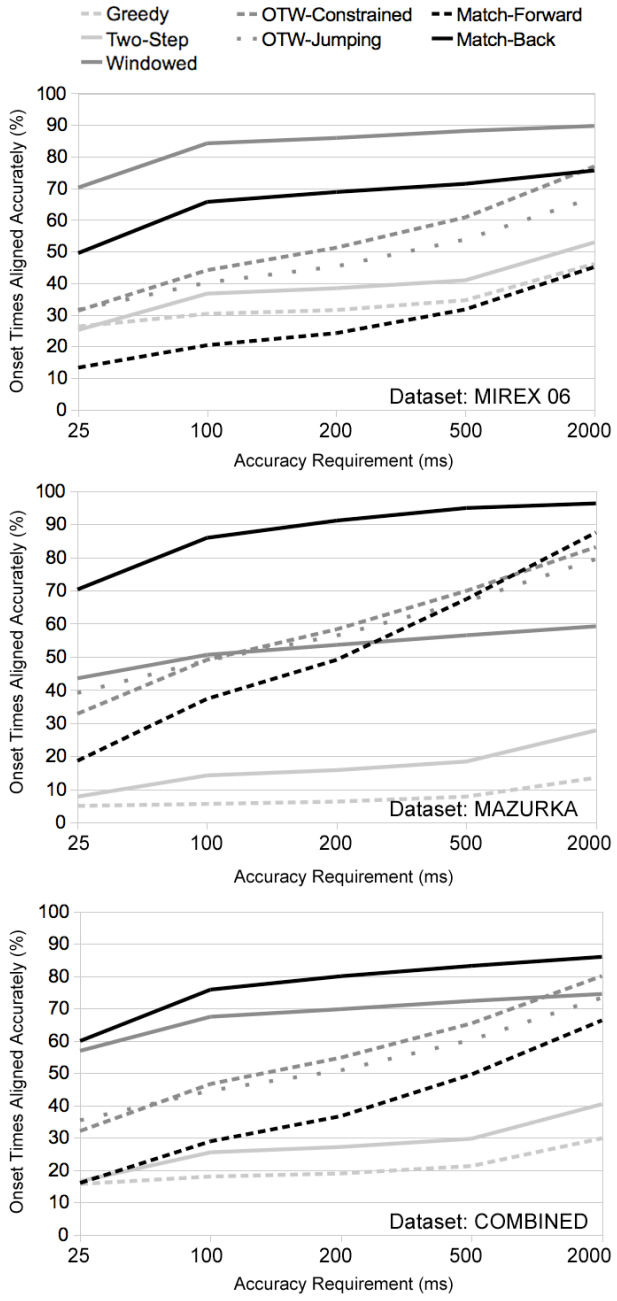


Figure 4.3: Accuracy rates of the methods evaluated for each dataset.

to calculate the cost matrix in a forward direction (OTW-Constrained, OTW-Jumping and Match-Forward) all follow a similar pattern across the various accuracy requirements where the proportion of correct alignments increases greatly for the less strict requirements. Conversely, the methods that use a backward calculation step (WTW and Match-Back) have a greater relative performance for the stricter requirements but this advantage diminishes for the less strict requirements. As such we can deduce that the backward based DTW paths provide an advantage in refining the initial paths to give more accurate score-aligned results, but are unable to correct for larger errors. Both Match methods perform better with the Mazurka data than with the MIREX 06 data, which is most likely because the Match methods use a different (multi-octave) feature that is geared more towards classical piano music than the varied instruments used in the MIREX 06 data. Of the other path strategies, the OTW-Constrained and OTW-Jumping methods show mid-range results whose closeness could be explained by their similar coverage of the similarity matrix. For situations that require zero latency, OTW-Constrained and OTW-Jumping offer the best return on accuracy. As expected, the weakest methods are the greedy and two-step path finding algorithms, which trade accuracy for speed.
Figure 4.4 shows a comparison of how the path search width affects 3 of the methods.

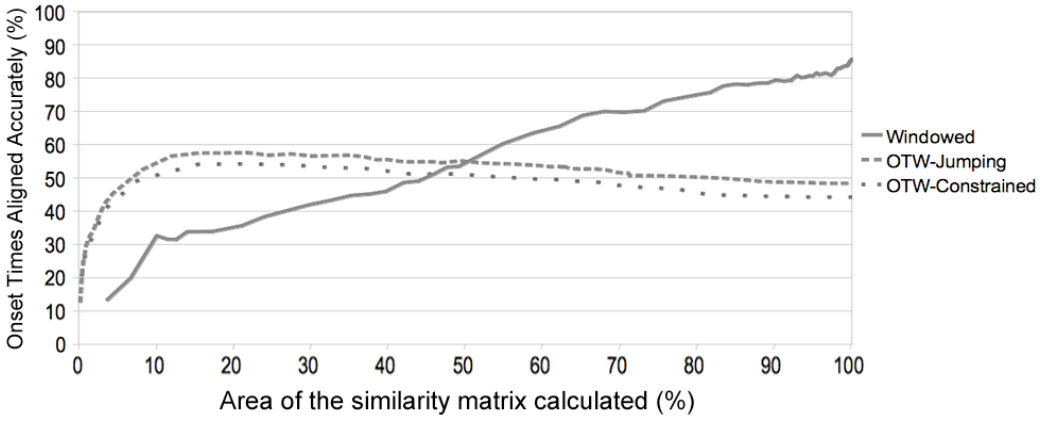


Figure 4.4: Onset accuracy rates (at the 200ms accuracy requirement) and the area of the similarity matrix covered at various bounding limits for three methods.

Figure 4.4 shows that larger bounding limits increase the area of the similarity matrix, with varying effects on the algorithms' accuracy. WTW has a straightforward correlation between the

area covered and the accuracy provided. However, for OTW-Jumping and OTW-Constrained, after a certain point, an increased search width can result in lower accuracies. This is because these methods do not utilise a typical DTW cost path and each path step calculation is based on the information from one row or column of the cost-matrix. Larger areas increase the chance of providing low cost alignment points far from the desired path and therefore can have a negative impact on the path found.

4.4.1 MIREX 2006 Comparison

Table 4.3 shows the accuracy rates for each method using the MIREX 06 data with an accuracy requirement of 2000 ms, as was used in the MIREX Score Following test in 2006. We also include the two methods evaluated that year, a Hidden Markov Model based approach with no Latency by Cont and Schwarz [2006] and a DP method by Puckette.[5] We can see that, of the real-time DTW based methods, Windowed and OTW-Constrained could be used in a score following application as they are comparable with the methods that were evaluated with the MIREX 06 data.

Piecewise Precision Rates							
Cont	Windowed	OTW-Constrained	Puckette	OTW-Jumping	Two-Step	Greedy	Match-F
90.1%	89.0%	76.4%	69.7%	66.6%	52.5%	45.6%	44.7%

Table 4.3: A comparison with the Score Following methods from MIREX 2006

4.5 Computational Efficiency

The computational efficiency tests consisted of aligning sequences of varying lengths and recording the execution time taken for the methods to make the alignment. The results of this test can be seen in Table 4.4. The methods tested are DTW and FastDTW from Salvador and Chan [2004] and examined in Section 2.2.4, and the six[6] real-time modifications of DTW proposed in Chapter 3. The results show quadratic time costs in the case of DTW. FastDTW has linear computational costs, however, in the case of the 100000 frame long sequences, FastDTW runs

[5] For more information on this evaluation see http://www.music-ir.org/mirex/wiki/2006:Score_Following_Results
[6] the sixth being the A-Star variation of WTW

out of random access memory and slows down as the search window is swapped in and out of ram. As such we can deduce that FastDTW has at least linear memory costs. Greedy, 2-Step, WTW, WTW A-Star, OTW-Jumping, and OTW-Constrained do not suffer from this slow down as their memory requirements are constant. As such we could increase the maximum sequence length to a million frames.

Execution time (seconds)					
Sequence length in frames	100	1000	10000	100000	1000000
Sequence length in hh:mm:ss	00:00:02	00:00:20	00:03:20	00:33:33	05:33:33
(presuming a 20ms hop size)					
DTW	0.02	0.92	57.45	7969.59	-
FastDTW (radius=0)	0.01	0.02	0.38	67.94	-
FastDTW (radius=100)	0.02	0.06	8.42	207.19	-
GREEDY	0.004	0.006	0.026	0.262	2.533
2-STEP	0.012	0.026	0.143	1.703	24.298
WTW	0.014	0.101	1.027	10.059	103.370
WTW A-Star	0.014	0.097	0.999	9.900	102.424
OTW-Jumping	0.011	0.064	0.667	6.567	67.006
OTW-Constrained	0.018	0.063	0.644	6.402	66.158

Table 4.4: Efficiency test results showing the execution time (in seconds) for 5 different lengths of input sequences (in frames). Results for the two FastDTW algorithms and DTW are from [Salvador and Chan, 2004]. WTW A-Star and WTW refer to Windowed Time Warping with and without the A-Star modification, respectively (see Section 3.4.4).

The 6 modifications from Chapter 3 are all faster than FastDTW with a search radius, but all except for the Greedy method are slower than FastDTW without a search radius (except when the sequence reaches more than 10000 frames). As expected, the Greedy method requires the least computation due to its minimalist approach. The 2-Step method takes approximately 5-10 times longer which is a large increase to add a minor second step to the algorithm. This increase is due to the computational complexity of storing subsequent path alignment steps in a linked list. OTW-Constrained and OTW-Jumping are the next most computationally expensive. OTW-Constrained is slightly more efficient than OTW-Jumping which reflects the

constrained path movement effect leading to a smoother path and therefore less of the cost matrix is calculated in the search radius. WTW and WTW A-Star are the most computationally expensive of the 6 new modifications due to the overlapping windows leading to repeated calculations of overlapping parts. The A-Star modification leads to a slight improvement in computational time.

4.6 Manual and Automatic Test data Comparison

Finally, in Figure 4.5, we compare the agreement rates between all the methods and the various reference data types.

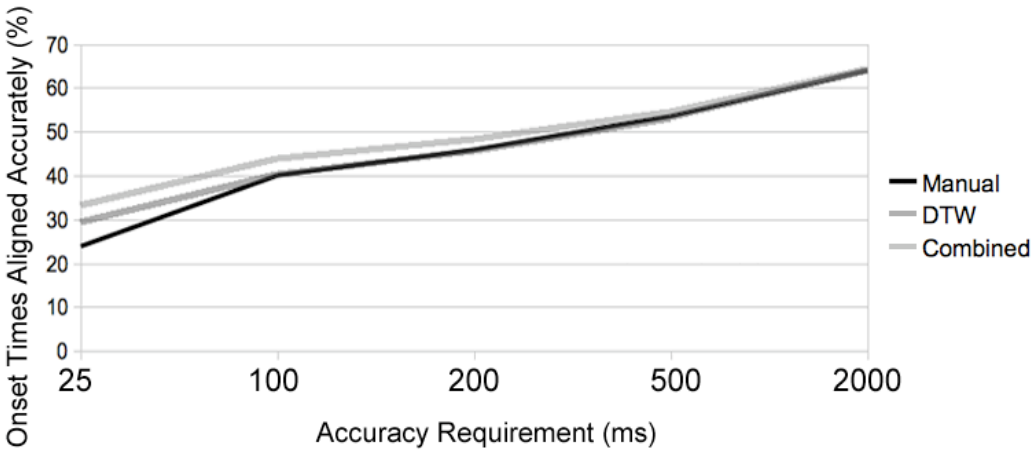


Figure 4.5: Average accuracy rates across all seven methods for the three datasets used.

Figure 4.5 shows that the alignment methods, when assessed at the 25 ms accuracy requirement, receive higher scores with the Combined reference data, an average of 3.7% higher than the DTW reference data and a further 5.6% over the Manual data (humans have trouble discriminating audio at this accuracy range). This could either be due to the greater accuracy achieved by combining references, or because the filtering out of "disputed" points leaves a subset of points which are easier to identify accurately. At the lower levels of accuracy, the methods perform equally with all three of the reference data types. These results suggest the possibility of using automatic techniques to refine manual annotations when high precision is

required. Such refinements would only be trusted when the disparity is small; during testing it was apparent that in some cases the off-line alignment failed completely, and thus we argue that manual referencing will always be required to validate automatically produced data.

4.7 Conclusions

In this chapter we have evaluated seven linear-time score following methods for music and scores using two datasets. The mixed results of the two datasets and modification factors such as local constraints demonstrate the importance of tailoring score following systems to their specific application. The Dynamic Programming methods from Chapter 3 have different balances of accuracy and response time so where one method with low latency, such as OTW-Constrained, might be suitable for applications requiring immediate feedback, another, such as the Greedy method, may be useful for large off-line database alignments. A significant feature of the new methods is in having constant memory costs, as demonstrated in Section 4.5. The capability of synchronising ever-longer sequences raises the possibility of feature film length sequences, or even continuous synchronisation.

Experimentation with local constraints has shown that the usual Type I and Type II cost constraints used are not always the most appropriate. We have also explored the relationship between accuracy rates and the proportion of the similarity matrix covered and found how parameter settings affect the alignment path accuracy.

Producing ground truth data for automatic alignment methods is a difficult and time consuming procedure. Methods proposed in this work, supported by a visual reference checking tool, allow large datasets to be checked in a semi-supervised manner. Whilst hand annotation is essential for grounding the data, automatic methods of generating annotations can identify onset times more precisely than human tapping, and a combination of the two can be used to correct errors in either method, such as missed notes or the failure of an alignment method, which would otherwise compromise the data. This method could lead to the production of larger and more accurate evaluation datasets for score following and alignment systems.

For larger scale datasets, hand annotations are not a feasible option. However, with the increasing availability of large amounts of data, it would be wasteful not to make use of this data in order to thoroughly test music and metadata linking algorithms encompassing a wide

variety of music styles. When automatic methods for the proposed task exist that produce near optimal solutions, hand annotations may not be necessary. Our evaluation shows a strong agreement between the manually and automatically produced sets of data and so we could theoretically extend the automatically produced dataset with a reasonable level of confidence in its accuracy.

This evaluation has shown that modifications of Dynamic Programming can be made to allow real-time alignment of sequences with comparable performances to state-of-the-art score following systems in both accuracy and efficiency. In the next chapter, we apply Dynamic Programming methods to assist in evaluating guitar tablature and chord sequences mined from the web.

5

Guitar Tab Information Retrieval

"you can find any song you can think of... and pull up the lyrics, the notes... its more accurate than [what] you buy in a bookstore, because these are the guitar nuts, they're sticklers about everything being exact" Saul Hudson[1] (aka Slash[2])

Contents

Music tablature has been around since the 14th century [Apel, 1961] and even online guitar tabs predate web browsers, having first been shared on USENET in the alt.guitar.tab forum. Now, with over 4.5 million tablatures and chord sequences (which we collectively refer to as tabs), the web holds vast quantities of hand annotated scores in non-standardised text. There are a number of digital music notation formats, such as Music XML, the MIDI file format, and various formats for images of scanned sheet music. However it is tabs, which are plain ASCII text files containing tablature and/or chord symbols and lyrics, that have become the most commonly used music notation format on the internet. A comparison of the most popular MIDI, sheet

[1] http://www.ultimate-guitar.tv/misc/slash_talks_about_ultimate_guitar.html
[2] Disclosure: Slash has a partnership with www.ultimate-guitar.com

File type	Most popular site	UVPM
tabs	ultimate-guitar.com	2,148,009
sheet music	8notes.com	538,071
MIDI	freemidi.org	10,258

Table 5.1: UVPM (Unique Visitors Per Month) to music score websites from http://siteanalytics.compete.com

music and tab websites' unique visitors per month can be seen in Table 5.1. The popularity of tabs is due to a simple, intuitive approach to the instructions that requires no formal training to understand nor specific software to read or write. Added to this is the fact that tabs are commonly free to use and the amount of data needed to transfer the text instructions is almost negligible. Other reasons as to why people contribute tabs are examined in [Chesney, 2004].

Due to the lack of requirements in contributing tabs and a lack of standardisation there are many variations in how tabs are structured, making machine parsing difficult. Also, as there are no restrictions on authorship of tabs, they are typically error-prone, incomplete, and tab collections contain many duplicates. All of these reasons makes retrieving high quality tabs difficult.

Prior to this work, tablature recognition was restricted to optical music recognition methods for reading printed tabs. Examples of Optical Tablature Recognition systems include [Dalitz and Karsten, 2005; Dalitz and Pranzas, 2009], and enhanced using HMMs [Pugin, 2006], and Fourier Descriptors [Wei et al., 2008]. In this chapter, we aim to decipher the many various formats of guitar tabs so that we can analyse and determine exactly how accurate online guitar tabs are, using web mining, text analysis, and Dynamic Programming to compare the tabs musical sequences.

In order to judge the accuracy of guitar tabs we define our ground truth data in Section 5.1. We explain how we crawl the web for guitar tabs in Section 5.2. In Section 5.3 we introduce our tab parsing system. We describe new functions for comparing guitar tabs chords and structure in Section 5.4. Using these functions, we evaluate guitar tab statistics in Section 5.5. We introduce the Guitar Tab Toolkit for applying these methods, as well as tab synthesis, in Section 5.6. Finally we conclude our work on guitar tab information retrieval in Section 5.7.

5.1 Ground Truth Dataset

In this work we focus on The Beatles due to the availability of ample annotated data and guitar tabs for this band. For assessing how accurate chord sequences are we use the ground truth chord sequence annotations for The Beatles from transcriptions by Harte et al. [2005]. This data includes chord sequences for 180 tracks from 12 Beatles' studio albums. An example of the chord annotations can be seen in Table 5.2. This dataset has been used within 50 papers and articles in MIR to date. For assessing how accurate structure segmentation is we use structure annotations by Mauch et al. [2009]. These are structural segmentations consisting of start time, end time and segment label for the 180 The Beatles tracks. The labels consist of words such as **verse, refrain, bridge, intro, outro,** and **silence**, which are often qualified with details, *e.g.***verse_a, verse_b,** and **verse_(guitar_solo)**. An example of the structure ground truth can be seen in Table 5.3.

Start	End	Chord	Start	End	Chord	Start	End	Chord
0	0.44	N	11.22	12.6	E:min	21.21	22.48	F#:dim/b7
0.44	1.01	D	12.6	13.2	E:min/b7	22.48	23.7	D
1.01	2.35	G	13.2	14.42	G	23.7	24.93	D/b7
2.35	3.58	D	14.42	15.67	D/3	24.93	26.15	G/3
3.58	6.13	G	15.67	16.98	E:min	26.15	26.76	D
6.13	6.79	C	16.98	17.57	E:min/b7	26.76	27.95	G
6.79	8.72	D	17.57	18.78	A:min	27.95	29.17	D/3
8.72	9.98	G	18.78	19.98	G	29.17	30.47	E:min
9.98	11.22	D/3	19.98	21.21	D/3	30.47	31.06	E:min/b7

Table 5.2: Initial 30 seconds of the chord ground truth chord annotation for The Beatles - All You Need Is Love.

5.2 Guitar Tab Mining

A typical web crawler, such as those used by search engines to examine web links, benefits from being able to identify its target easily. Website addresses not only have a distinct format but are wrapped in a standard "<a href=address>" HTML tag. As such, web crawlers require

Start	End	Structure Label
0.000	0.400	silence
0.400	26.773	intro
26.773	44.262	verse
44.262	61.581	verse
61.581	79.219	refrain
79.219	96.068	verse_(guitar_solo)
96.068	113.459	refrain
113.459	130.091	verse
130.091	147.170	refrain
147.170	164.257	refrain
164.257	224.000	outro_(fade-out)
224.000	228.440	silence

Table 5.3: Structural ground truth annotation for The Beatles - All You Need Is Love.

one set up in order to be able to crawl any standard HTML web page. When mining the web for guitar tabs we do not have this luxury and a page containing a guitar tab is not easily identifiable. A guitar tab web crawler must therefore know each domain's structure in order to retrieve the addresses for the tabs on that domain. It is not easy to say exactly how many guitar tab domains exist but in our studies we have encountered 286 at the time of writing. Rather than writing a crawler for each of them, we instead mine the search engines which directly link to the tabs within these domains. For this we use two search engines that are typically used when looking for tabs, Google.com and the more specialised 911tabs.com.

5.2.1 Google

For each of the 180 Beatles tracks in our ground truth dataset, we searched Google for the top 100 ranked guitar tabs. In order to ensure the results contained a high ratio of tabs in the results, we used a combination of search terms ("guitar", "tab" and some filters for unwanted content such as "-video") combined with the artist and track name. Links returned that are not tabs are not so much a concern as they will become apparent when parsing the web page for guitar tab content.

5.2.2 911tabs

911tabs.com is a guitar tab search engine with over 4.5 million tabs indexed. We wrote a web crawler that browsed 911tabs collection for "The Beatles" and gathered the links for any guitar tabs corresponding to the 180 tracks in our test set.

In total we found 24746 tabs relating to the 180 Beatles tracks in our ground truth data.

5.3 Guitar Tab Parsing

We view decoding tabs as an example of noisy text analytics, which are often applied to determine meaning from web mined resources such as online chat, forums, blogs and wikis. To interpret the noisy semi-structured tab data, we implemented a large set of simple heuristics to handle the many varied tab writing styles that exist. The following steps are a brief outline of the stages involved in parsing tabs.

- Define a pre-set chord dictionary of 410 unique chords from ultimate-guitar.com[3]. The list of chords used, along with their alternative names, notes and an example fret numbering can be found in Appendix A.
- Interpret any HyperText Markup Language (HTML) specific tags. For instance,   and
tags are changed to spaces and new lines, respectively.
- Analyse each line to determine what (if any) type of tab line it is. For example the line could contain a structural marker, chords, lyrics, tablature, *etc.* Non-tab-specific text is discarded at this point. Explained in Section 5.3.1.
- Structurally segment the tab based on any structural markers such as *intro*, *chorus*, *solo*, *etc.* Explained in Section 5.3.2.
- Analyse the ordering of tab line types that correspond to the same part of the song and group these tab lines into "systems". Explained in Section 5.3.3.

[3]Available at http://www.ultimate-guitar.com/lessons/chords/all_guitar_chords.html

- For each tab line, decode the tab elements according to the identified tab type. Explained in Section 5.3.4.
- Reorganise the tab systems according to any given structural information. Explained in Section 5.3.5.

5.3.1 Determining Tab Line Type

Once the HTML tags have been stripped from the text, a great deal of non-viewable page code still needs to be filtered out. The contents of the tags cannot be discarded as in some cases the entire tab is inside an HTML tag. To filter out non-relevant text we collected a set of indicators of code which we developed incrementally whilst building the parser using a test driven approach. Additionally if a line's proximity to another clean tab line is too far it is discarded, to ensure lines looking like tab lines surrounding the tab were not included. Such false positives might appear in user comments or if the page contained snippets from another tab. For the remaining identified tab content, we analysed each line to see if it fitted into one of the following categories in this order: *empty*, *capo_declaration*, *chord_definition*, *tuning_declaration*, *structure_layout*, *chord_definition_structural_indicator*, *structural_marker*, *tablature*, *chords*, *chords_and_lyrics*, *stroke_pattern*, and *lyrics*.

Each of these tab line types had its own set of heuristics. For example, a line was determined to be a *capo_declaration* line if it included the strings "capo" and "fret". Other heuristics are dependent on a number of sub-heuristics. For example to determine if a line has chords, we first define a set of rules to decide if a word is a chord. A word is considered a chord if it is less than 10 characters long, the first character is a note name, any three letter sequence is in the following list {"min","add","aug","dim","maj","sus","flat"}, and there are no four digit sequences (which would more likely indicate a chord definition). If a line has only chords we classify it as *chords*. If a line has chords and other characters that are letters we classify it as *chords_and_lyrics*.

5.3.2 Segmenting the Tab

Once the lines in a page have been divided into the tab line categories we analyse the structure of the tab. To begin with we look for guitar tab preliminaries such as *capo_declaration*, *tun-*

ing_declaration, *structure_layout*, and *chord_definition* lines. We then look for the start and end of the tab content which is often determined by the first and last *structural_marker*, *tablature*, *chords*, *chords_and_lyrics*, *stroke_pattern*, or *lyrics* line. This tab content is segmented based on preceding *structural_marker* lines containing structural labels like "intro", "verse", "chorus", "bridge" *etc.* When none are found, or the first segment has no preceding *structural_marker* line, an "unknown" segment is applied. Otherwise, the *structural_marker* is used as the label for the segment. After the last tab content line we then look for any further *structure_layout* or *chord_definition* lines which can also be found at the end of a tab.

5.3.3 Tab System Line Ordering

We define a tab system as a combination of tab line types that indicate a sequence of music where the chords, tablature notation and lyrics are in sync vertically. As an example, a common tab system line ordering would be: *chords*, *tablature*, *lyrics*. Tab Segment Sample 1 in Figure 5.1 shows a tab system line ordering of *chord_and_lyrics* and Tab Segment Sample 2 in Figure 5.2 shows a tab system line ordering of *chords*, *tablature*, *lyrics*. Tab Segment Sample 3 in Figure 5.3 shows tab system line orderings of *tablature* in Riff1, *tablature*, *chords* in Riff2 and *chords*, *lyrics* in the subsequent segment. Often the line proximity can decide the tab system grouping, with *empty* lines splitting the combinations. When there is no difference in the proximity, we use the initial tab system line ordering and when the first tab line type is found a new tab system is started. If the first tab line type is *lyrics* or *tablature* then we split bars using the *empty* lines as multiple lines of these types are often part of the same bar.

5.3.4 Decoding the Elements

Decoding the tab line types involves interpreting which fret the *capo_declaration* line is indicating the capo should be placed on, which tuning the *tuning_declaration* is indicating, *etc.* Chord labels and notes are read from *chord_definition* lines and if they are not already present in the chord dictionary they are added. Chord labels are then extracted from any *chords* and *chord_and_lyrics* lines and notes are extracted from *tablature* lines. The notes of the chord can now be looked up in the Chord Dictionary. If a direct match for the chord label is not found, common alternatives, such as removing any punctuation symbols from the chord label

(:,/,*,+), are tried instead. If the Chord Dictionary does not contain a chord corresponding to any variation of the chord label then an estimate of the chord is made by taking the closest matching chord label instead.

5.3.5 Tab Restructuring

After the tab content has been decoded, the tab is restructured to fit any structural edits or layouts found in the tab. Any indicators of repetitions will be expanded so that "x2" will result in the current section being duplicated. If a *structure_layout* line was found then the structure is matched to the layout described. This step is often required as tabs often show a section only once and leave the user to repeat the section as and when necessary. In order to compare guitar tabs by aligning their musical sequences, we need to replicate this reconstruction.

5.3.6 Parsing Evaluation

We evaluated our heuristics for parsing guitar tabs on a set of 20 tabs for which we manually annotated the ground truth data. The chord retrieval test set included 20 tabs with 880 chords, all of which were detected. Figures 5.1, 5.2, and 5.3 exemplify how tabs differ in format between each other and even within themselves.

```
A taste of [Am]honey, [C]tasting much [G]sweeter than [Am]wine

I [Am]dream of [C]your first [G7]kiss and [D]then
I [Am]feel a[C]part, my [G7]lips are [D]gett'n
A taste of [Am]honey, [C]tasting much [G]sweeter than [Am]wine

{Chorus:}
I [A]will re[C]turn, yes [D]I will re[Em]turn
I'll come [F]back for the [G]honey and [Am]you.
```

Figure 5.1: Tab Segment Sample 1. The Beatles - A Taste of Honey
Chords: Am C G Am Am C G7 D Am C G7 D Am C G Am A C D Em F G Am

```
  3/4 G       Am7     G/B     4/4 G
E{|---------------------------|-----------------------------------|
B{|---0-------1-------3-------|---12----12--12----12----12--12----|
G{|-------0-------0-------0---|-------0-------0-------0-------0---|
D{|---------------------------|-----------------------------------|
A{|-----------0-------2-------|---10------10------10------10------|
E{|---3-----------------------|-----------------------------------|
      +   .   +   .   +   .       +   .   +   .   +   .   +   .

  3/4 G       Am7     G/B     4/4 G
E||---------------------------|-----------------------------------|
B||---0-------1-------3-------|---12----12--12----12----12--12----|
G||.------0-------0-------0---|-------0-------0-------0-------0---|
D||.---------------------------|-----------------------------------|
A||-----------0-------2-------|---10------10------10------10------|
E||---3-----------------------|-----------------------------------|
      +   .   +   .   +   .       +   .   +   .   +   .   +   .
   1. Blackbird singing in the dead of night
   2. Blackbird singing in the dead of night

      C       A7/C#   D       B7/D#       Em              Cm/Eb
E |-----------3---------------5-------|-----------------------------------|
B |---5--(5)----------7--(7)----------|---8-----8---8-----8-----8---8-----|
G |-------0-------0-------0-------0---|-------0-------0-------0-------0---|
D |-----------------------------------|-----------------------------------|
A |---3-------4-------5-------6-------|---7-------7-------6-------6-------|
E |-----------------------------------|-----------------------------------|
      +   .   +   .   +   .   +   .       +   .   +   .   +   .   +   .
      Take these broken wings and learn to fly
      Take these sunken eyes and learn to see (notes in () not played 2nd verse)
```

Figure 5.2: Tab Segment Sample 2. The Beatles - Blackbird
Chords: G Am7 G/B G G Am7 G/B G C A7/C# D B7/D# Em Cm/Eb

```
[Intro]
Riff1
e------0-|-3---3---5---5-|-10-------------------|------8---------|
B---3--1-|-3---3---7---7-|-12----12-0--0--12-0--|------10--10--7-|
G--------|-4---4---7---7-|-12----12-12-12-12-12-|-9----9---------|
D--------|---------------|----------12-12-12-12-|-10------------|
A--------|---------------|----------------------|(10)------------|
E--------|---------------|----------------------|----------------|

Riff2
e--3--3--3--3--|--0--0------(0)-|
B--3--3--3--3--|--3--3--(3)-----|
G--0--0--2--2--|--0--0----------|
D--0--0--0--0--|--2--2---0---2--|
A--2--2--x--x--|--2--2----------|
E--3--3--2--2--|--0--0----------|
   G     D/F#     Em

G       D/F#    Em
Love    love    love
G       D/F#    Em
Love    love    love
D7/A    G       D7/F#    D7/E
Love    love    love
D    C    Riff3
```

Figure 5.3: Tab Segment Sample 3. The Beatles - All You Need Is Love
Chords: G D/F# Em G D/F# Em G D/F# Em D7/A G D7/F# D7/E D C

5.4 Comparing Guitar Tabs

In order to judge the accuracy of guitar tabs we need a metric with which to compare them with ground truth chord and structure sequences. There already exist methods for comparing chord sequences that are used in the MIREX: Audio Chord Detection/Estimation task [4], however these methods rely on timing information associated with the chord sequences. As there is only a rough indication of timing/rhythm information in online guitar tabs, these methods are unsuitable in this case. Our solution is to use DP to measure the similarity between the tab's chords, chord sequence, and structure sequence, with that of the ground truth.

[4] http://www.music-ir.org/mirex/wiki/2011:Audio_Chord_Estimation

5.4.1 Similarity Score

When using distance metrics such as the LED, or DTW, we often refer to a similarity score, calculated by summing the costs along the optimal alignment path. This cost is related to the length of the input sequences $U = (u_1, u_2, ..., u_M)$ and $V = (v_1, v_2, ..., v_N)$ in that it will never exceed the maximum length of the two, *i.e.* $DC <= \max(M, N)$ (assuming the maximum cost of each step is 1). The problem with this cost metric is that 2 long sequences will inevitably have a higher cost than two shorter sequences such as in LED ("ABBABACADABRA","BABBACOMBE") = 6, and LED ("ABBA","XYZ") = 4. Instead, we propose a "Similarity Score" whereby the cost is normalised relative to the perfect score and inverted then multiplied by 100 to give a similarity score from 0 to 100: $SimilarityScore = \left(1 - \frac{\mathrm{DP}(U,V)}{\max(M,N)}\right) \times 100$. Using this metric, the LED "score" for ("$ABBABACADABRA''$, "$BABBACOMBE''$) = 54%, whereas ("$ABBA''$, "XYZ'') = 0%. We find the similarity score is a more useful metric for comparing sequences.

5.4.2 Chord Difference (CD)

The Chord Difference measures the difference of two chords. The intention of the Chord Difference is to calculate a substitution matrix, similar to that in the Needleman-Wunsch algorithm, for chords in order to align sequences of chords later. In order to measure the CD, we use the LED of the ordered pitch classes in the chords, as defined in the chord dictionary or interpreted from the chord definitions (when present). The notes are ordered alphabetically, starting from the first matching note in both sequences with preceding notes being appended to the end. As such, when measuring the CD of C# = (C#, G#, F) and C#6 = (A#, C#, F, G#) the set of notes are re-ordered as (C#, F, G#) and (C#, F, G#, A#). Therefore the CD uses DP to find a path $P(U, V) = (p_1, p_2, ..., p_W)$ through the matrix of costs between sequences $U = (u_1, u_2, ..., u_M)$ and $V = (v_1, v_2, ..., v_N)$. This cost matrix is described as $d_{U,V}(m, n)$ where $m \in [1 : M]$ and $n \in [1 : N]$ where each $p_k = (m_k, n_k)$. CD, like LED, uses a cost of 0 for matches and 1 for any insertion, deletion or alteration. The maximum cost is the length of the longest sequence $\max(M, N)$. We normalise the distance cost by dividing the LED of two

chords (note sequences), U and V by the maximum length.

$$\mathrm{CD}(U,V) = \left(\frac{\mathrm{LED}(U,V)}{\max(length(M), length(N))}\right) \tag{5.1}$$

An example of the CD function being calculated and results from this function can be seen in Tables 5.4 and 5.5, respectively.

		Dm/C#			
		C#	D	F	A
	C#	**0**	**1**	2	3
C#	F	1	1	**1**	2
	G#	2	2	2	**2**

Table 5.4: Chord Difference (CD) example: $\mathbf{2}/4 = 0.5$

	C#	C#6	Db	Fm7	C/B	A	D
	C#,F,G#	A#,C#,F,G#	C#,F,G#	C,D#,F,G#	B,C,E,G	A,C#,E	A,D,F
C#	0.0	0.25	0.0	0.5	1.0	0.67	1.0

Table 5.5: Chord Difference (CD) examples.

5.4.3 Chord Sequence Difference (CSD)

To calculate the difference between two sequences of chords T_1 and T_2, we use DTW, which has been used for synchronisation in applications such as score following [Dannenberg, 1984]. Unlike the binary cost function in LED, DTW can use a more detailed cost function such as the inner product of the pair of feature vectors, which returns a value between 0 and 1 for each pair of feature vectors. In our case the DTW uses the CD cost function to compare chords. Additionally, we remove any duplicates of a chord from the chord sequences to avoid penalising tabs that specify a chord is to be played twice in succession. The overall path alignment cost is given by the sum of the individual chord match costs along the DTW path P:

$$\mathrm{CSD}(T_1,T_2) = \mathrm{DTW}(T_1,T_2,CD) \tag{5.2}$$

5.4.4 Chord Sequence Similarity (CSS)

The Chord Sequence Similarity is a measure of how similar two tab chord sequences, T_1 and T_2 are. As the maximum possible cost is the length of the longest sequence, we convert the CSD into a normalised similarity score, expressed as a percentage:

$$\mathrm{CSS}(T_1, T_2) = \left(1 - \frac{\mathrm{CSD}(T_1, T_2)}{\max(length(T_1), length(T_2))}\right) \times 100 \tag{5.3}$$

An example of the CSS being calculated and results from this function can be seen in Tables 5.6 and 5.8, respectively.

	E	B7	E	F#7	B7	E7	A	E7	B7	A	F#7	B7	E	C#7	F#7	B7	E
E	**00.0**	00.8	00.8	01.5	02.3	02.5	03.5	03.8	04.5	05.5	06.3	07.0	07.0	07.5	08.3	09.0	09.0
F#	01.0	**00.8**	01.8	01.0	01.8	02.8	03.2	04.2	04.5	05.2	05.4	06.2	07.2	07.8	07.8	08.5	09.5
B	01.7	**01.0**	01.4	01.8	01.3	02.0	03.0	03.8	04.0	05.0	05.8	05.7	06.3	07.1	07.8	08.0	08.7
E	01.7	01.8	**01.0**	01.8	02.0	01.5	02.5	02.8	03.5	04.5	05.3	06.0	05.7	06.2	06.9	07.7	07.7
C#	02.3	02.7	**01.7**	01.8	02.8	02.3	02.5	03.3	03.8	04.5	05.3	06.3	06.3	05.9	06.7	07.7	08.3
F#	03.3	03.1	02.7	**01.9**	02.5	03.3	02.9	03.5	04.0	04.4	04.7	05.4	06.4	06.7	06.2	06.9	07.9
B	04.0	03.3	03.3	02.7	**02.2**	02.9	03.9	03.7	03.8	04.8	05.2	04.9	05.6	06.3	06.9	06.4	07.1
E	04.0	04.1	03.3	03.4	02.9	**02.4**	03.4	03.7	04.4	04.8	05.5	05.7	04.9	05.4	06.2	06.9	06.4
D	05.0	04.5	04.3	04.1	03.4	03.2	**03.1**	03.8	04.2	04.8	05.5	06.0	05.9	05.9	06.2	06.7	07.4
E	05.0	05.3	04.3	04.8	04.2	03.4	04.1	**03.3**	**04.1**	05.1	05.6	06.3	05.9	06.4	06.7	06.9	06.7
F#	06.0	05.8	05.3	04.6	04.9	04.4	04.1	04.3	04.1	**04.8**	**05.0**	05.8	06.8	06.7	06.7	07.4	07.7
B	06.7	06.0	06.0	05.3	04.8	05.2	05.1	04.8	04.3	05.1	05.5	**05.3**	05.9	06.7	07.4	06.9	07.6
E	06.7	06.8	06.0	06.1	05.6	05.1	06.1	05.1	05.1	05.3	05.8	06.0	**05.3**	05.8	06.5	07.3	06.9
C#	07.3	07.7	06.7	06.8	06.6	05.8	06.1	05.8	06.1	06.1	06.1	06.8	05.9	**05.5**	06.3	07.3	07.6
F#	08.3	08.1	07.7	06.9	07.3	06.8	06.5	06.8	06.6	06.8	06.3	06.8	06.9	06.3	**05.8**	06.5	07.5
B	09.0	08.3	08.3	07.7	07.2	07.6	07.5	07.3	06.8	07.6	07.1	06.6	07.3	07.0	06.5	**06.0**	06.7
E	09.0	09.1	08.3	08.4	07.9	07.4	08.4	07.5	07.6	07.8	07.8	07.3	06.6	07.1	07.3	06.8	**06.0**
D	10.0	09.5	09.3	09.1	08.4	08.2	08.1	08.3	08.0	08.3	08.6	07.8	07.6	07.6	07.8	07.3	**07.0**

Table 5.6: Chord Sequence Similarity (CSS) example: (1-(**7.0**/18))*100.0 = 61.1%

E	B7		E		F#7	B7	E7	A	E7	B7	A	F#7	B7	E	C#7	F#7	B7	E	
E	F#	B	E	C#	F#	B	E	D	E		F#		B	E	C#	F#	B	E	D

Table 5.7: Alignment of the two chord sequences from the example in Table 5.6.

Sequence 1:	Bb Eb7 Bb F9 E Bb F Bb Eb7 Bb F9 E Bb Bb7 Eb Bb Eb F Bb Eb7 Bb F9 E Bb Bb7 Eb Bb Eb F Bb Eb7 Bb F9 E Bb F Bb Eb Ebm
Sequence 2:	Bb Eb Bb F Eb Bb F Bb Eb Bb F Eb Bb Bb7 Eb Bb Bb7 Eb F Bb Eb Bb F Eb Bb Bb7 Eb Bb Bb7 Eb F Bb Eb Bb F Eb Bb Bb7 Bb Eb Ebm
	Chord Sequence Similarity: 74.9%
Sequence 1:	D D7 G D7 A G D7 G D7 A G D7 G D7 A G D7
Sequence 2:	D D7 G D A G D7 G D A G D7 D G G7 D A G D
	Chord Sequence Similarity: 92.9%
Sequence 1:	E A E B E A E B E A E B E A E B E A B A E A E B E A E B E B E A E B E A B A E A E B E A E B E B E B A E E9
Sequence 2:	A D7 A E7 A A7 D F A E7 A D7 A E7 A A7 D F A E7 A D7 E D A D7 A E7 A A7 D F A E7 A D7 E D A D7 A E7 A A7 D F A E7 A E7 A E7 A
	Chord Sequence Similarity: 55.5%
Sequence 1:	E F# B E C# F# B E D E F# B E C# F# B E D E F# B E C# F# B E D E
Sequence 2:	C G7 C G7 C G7 C G7 C F C G7 C G7 C Am(add9) G7 C G7 C F G7 C G7 C G7 C F
	Chord Sequence Similarity: 22.8%
Sequence 1:	G D7 C7 G D G D7 C7 G D G Bm A A7 D Em A D G D7 C7 G D G Bm A D Em A D G D7 C7 G D G
Sequence 2:	G C9 G C9 G C9 G C9 G C9 G C9 D C7 G D G Bm A D E7 A7 D7
	Chord Sequence Similarity: 60.1%
Sequence 1:	Am7 Dm G7 C E Am Dm9 G7 C Fmaj7 C Fmaj7 C E Am Dm9 G7 C Am7 Dm G7 C E Am Dm9 G7 C
Sequence 2:	Am7 Dm7 G7 C Em Am Dm9 G7 C F C F C Em Am Dm9 G7 C Am7 Dm7 G7 C Em Am Dm9 F
	Chord Sequence Similarity: 87.8%

Table 5.8: Chord Sequence Similarity (CSS) examples.

5.4.5 Chord Accuracy (CA)

The Chord Accuracy measures the similarity of the sequence of chords T in a tab to the chord sequence G in the ground truth data for the song. Transpositions are not considered in this factor.

$$\mathrm{CA}(T, G) = \mathrm{CSS}(T, G) \tag{5.4}$$

5.4.6 Segment Chord Accuracy (SCA)

The Segment Chord Accuracy is an alternative accuracy measure for sequences of chords. Many tabs have incomplete chord sequences, and rely on the user to piece together the complete tab based on cues, intuition and knowledge of the song. Such tabs receive a low score when evaluated with CA against the ground truth. A more flexible accuracy measurement, the Segment Chord Accuracy, finds the accuracy of each segment in the tab independently. For each segment of a song, as defined in our structural ground truth data, the SCA takes the closest matching sub-sequence from the tab's overall chord sequence using the CSD. In addition, chord sub-sequences which match to more than one segment may be reused and transpositions of the overall chord sequence are allowed in the SCA. This is because many tabs have few repeated sections and can be written in an alternative key to the original music. The SCA returns a normalised similarity score. The pseudo-code for the SCA is shown in Algorithm 8.

```
Input: Segmentation S = {s_1, s_2, ..., s_l}, Ground Truth Chords G, Tab Chords T
Output: Segment Chord Accuracy SCA
SCA = length(G);
for Transposition Tr = 0 to 11 do
    TranspositionCost = 0;
    for i = 1 to l do
        SegCost = length(s_i);
        for start = 0 to length(T) do
            for len = 1 to length(T) − start do
                T' = subsequence(T,start,len)
                if CSD(s_i, T') < SegCost then
                    SegCost = CSD(s_i, transpose(T', Tr));
                end
            end
        end
        TranspositionCost += SegCost ;
    end
    if TranspositionCost < SCA then
        SCA = TranspositionCost;
    end
end
return (1 − SCA)/ max(length(S), length(G))) × 100;
```

Algorithm 8: Segment Chord Accuracy

5.4.7 Structure Similarity (SS)

In order to calculate Structure Similarity we first normalise the labelling of structural segments, so that a musical structure such as (Intro, Verse, Chorus, Verse,...) is represented by the sequence of characters (A, B, C, B, ...). To convert the labels into the character sequence we define a list of known labels. For each structure label encountered, if the first half of the label is present in the known list of labels we re-use the character for that label, otherwise a new character is assigned and the label is added to the list. In this manner, Verse 1 and Verse 2 are treated as the same symbol. We then propose using the LED to calculate the cost between these sequences of characters. We convert the LED to a similarity score:

$$\mathrm{SS}(T_1, T_2) = \left(1 - \frac{\mathrm{LED}(T_1, T_2)}{\max(length(T_1), length(T_2))}\right) \times 100 \tag{5.5}$$

Note that we only compute Structure Similarity where the structure is explicitly given in the tab. An example of the SS being calculated using this function can be seen in Table 5.9. Table 5.9 shows examples of the SS on sample structure sequences and an example of an alternative labelling sequence can be seen in the first example in Table 6.7.

Sequence 1: intro, verse, verse, bridge, verse_, verse_(solo), bridge, verse-v2, outro
Sequence 2: unknown, verse 1, verse 2, middle 1, verse 1, solo, middle 1, verse 1

	A	B	B	C	B	B	C	B	D
A	**0**	1	2	3	4	5	6	7	8
B	1	**0**	0	1	1	1	2	2	3
B	2	0	**0**	1	1	1	2	2	3
C	3	1	1	**0**	1	2	1	2	3
B	4	1	1	1	**0**	0	1	1	2
D	5	2	2	2	1	**1**	1	2	1
C	6	3	3	2	2	2	**1**	2	2
B	7	3	3	3	2	2	2	**1**	**2**

Table 5.9: Structure Similarity (SS) example: (1-(**2**/9))*100.0 = 77.8%.

Sequence 1: refrain, verse, refrain, verse-v3, refrain, bridge, refrain-v2, outro (simplified) A B A B A C A D Sequence 2: chorus, verse 1, verse 2, chorus, verse 3, verse 4, chorus, solo, chorus (simplified) A B B A B B A C A Structure Similarity: 77.8%
Sequence 1: intro, verse, verse, bridge, verse, bridge, verse, outro (simplified) A B B C B C B D Sequence 2: intro, verse 1, verse 2, bridge 1, verse 3, bridge 2, verse 4 (simplified) A B B C B C B Structure Similarity: 87.5%
Sequence 1: intro, refrain, verse, refrain, verse, refrain-v2, bridge, refrain_with_ending (simplified) A B C B C B D E Sequence 2: intro, verse, verse, bridge, verse, bridge, outro (simplified) A B B C B C D Structure Similarity: 75.0%
Sequence 1: intro, verse, verse, bridge, verse, bridge, verse, outro (simplified): A B B C B C B D Sequence 2: verse, verse, bridge, verse, bridge, verse, outro (simplified): A A B A B A C Structure Similarity: 50.0%

Table 5.10: Structure Similarity (SS) examples.

5.4.8 Structure Accuracy (SA)

The Structure Accuracy is a measure of how similar the structural sequence T of a tab is to the structural sequence G of the ground truth data.

$$\mathrm{SA}(T, G) = \mathrm{SS}(T, G) \tag{5.6}$$

5.4.9 Alternative Structure Accuracy (ASA)

The Alternative Structure Accuracy measurement allows for alternative labelling systems. It does this by removing elements from the sequence with the most labels and transposing the sequence of characters so that (A, B, C, B, ...) can also be represented as (X, A, B, A, ...). The sequence with the most labels is determined by counting the number of unique labels in both sequences. The difference in unique label counts d, represents the maximum number of

prior label segments we need to attempt removing from the sequence with the larger number of unique labels. For each of these prior labels, $1, 2, .., d$ we remove the first label and re-calculate the simplified sequence based on the new ordering of labels with a null label attached to the start and apply the LED between this modified sequence and the sequence with the smaller number of unique labels. Of all the possible label arrangements, we take the minimum LED as the Structure Similarity, converted to a distance score, as in Algorithm 9. The "re_sequence" function replaces the first non-"X" character with "X" and re-assigns the subsequent labels accordingly such as in the example given earlier. Table 5.11 shows the difference between the SA and the ASA on the last two examples from Table 5.9.

Input: *Structure Sequence* $A = \{a_1, a_2, ..., a_m\}$, *Structure Sequence* $B = \{b_1, b_2, ..., b_n\}$
(A being the Structure Sequence with the most unique labels)
Output: *Alternative Structure Accuracy ASA*
ASA = LED(A,B);
$d = unique\_labels(A) - unique\_labels(B)$;
for *Alternative Sequence* $alt = 1$ *to* d **do**
  $mod = re\_sequence(A, alt)$;
  if $LED(mod, B) < ASA$ **then**
    $ASA = LED(mod, B)$;
  end
end
return $1 - (ASA/\max(length(A), length(B))) \times 100$;

Algorithm 9: Alternative Structure Accuracy

5.5 Tab Statistics

Of the 24746 tabs found with our web mining, 7541 had recognisable chord content and 4643 had structure explicitly defined, with at least 3 chords/sections. The small proportion of tabs with chord/structure content is partly due to a number of the tab links being broken or simply not in text format, within the content of the web page found. The average tab CA, SCA, SA, and ASA for tabs, tabs that were duplicates, and tabs that were non-duplicates, can be seen in Table 5.12. It is interesting to note that duplicate tabs have comparatively higher chord accuracies (CA and SCA) and structure accuracies (SA and ASA) than non-duplicates. This suggests that accurate tabs are more likely to be copied. Figures 5.4, 5.5, 5.6, and 5.7 show histograms of the number of tabs and their spread of chord/structure accuracies.

Sequence 1: intro, refrain, verse, refrain, verse, refrain-v2, bridge, refrain_with_ending
(simplified) A B C B C B D E
Sequence 2: intro, verse, verse, bridge, verse, bridge, outro
(simplified) A B B C B C D
Structure Accuracy: 75.0%
Alternative Structure Accuracy: 75.0%
Sequence 1: intro, verse, verse, bridge, verse, bridge, verse, outro
(simplified): A B B C B C B D
(converted to): X A A B A B A C
Sequence 2: verse, verse, bridge, verse, bridge, verse, outro
(simplified): A A B A B A C
Structure Accuracy: 50.0%
Alternative Structure Accuracy: 87.5%

Table 5.11: Structure Accuracy (SA) and Alternative Structure Accuracy (ASA) examples.

	All	Duplicates	Non-duplicates	Samples
Chord Accuracy (CA)	63.1%	64.6%	59.8%	7541
Segment Chord Accuracy (SCA)	74.0%	75.7%	70.2%	7541
Structure Accuracy (SA)	52.2%	52.6%	51.3%	4643
Alternative Structure Accuracy (ASA)	58.8%	58.9%	58.5%	4643

Table 5.12: Accuracy rates for the guitar tabs of The Beatles. 69.6% of the chord containing tabs and 70.8% of the structure defining tabs were duplicates of other tabs.

5.5.1 Tab Domain Comparison

In this comparison we calculate the average accuracy of tabs for both chords and structure for different tab domains. We select only the domains which have over a hundred tabs of The Beatles with chords or structure detected. The top ten for Chord Accuracy are shown in Table 5.13.a and the top ten for Structure Accuracy are shown in Table 5.13.b. The tab count in the last column shows how many tabs for The Beatles were found on that domain with chord/structure content.

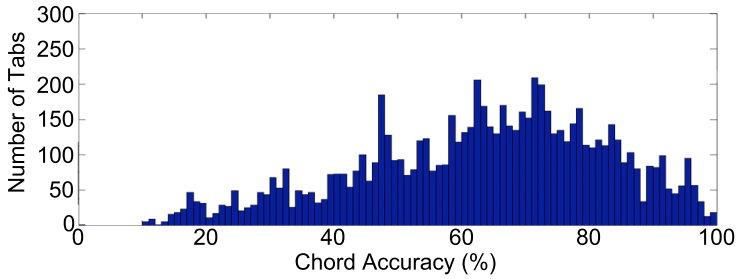


Figure 5.4: A histogram of the guitar tabs CA.

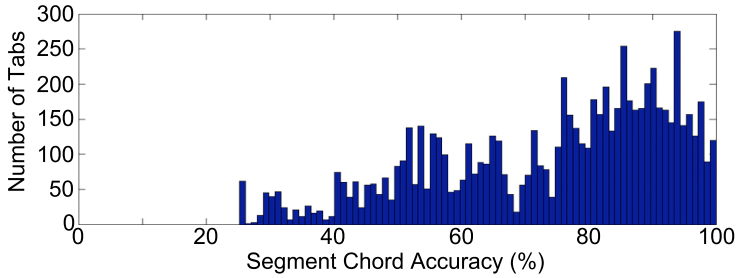


Figure 5.5: A histogram of the guitar tabs SCA.

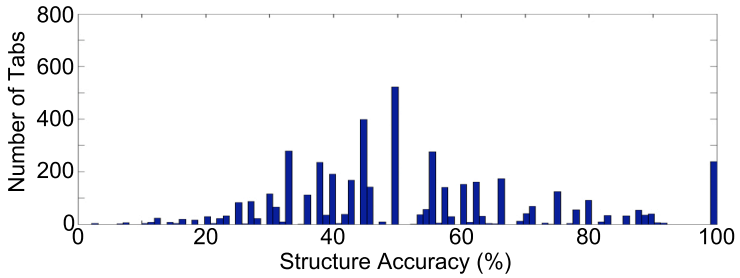


Figure 5.6: A histogram of the guitar tabs SA.

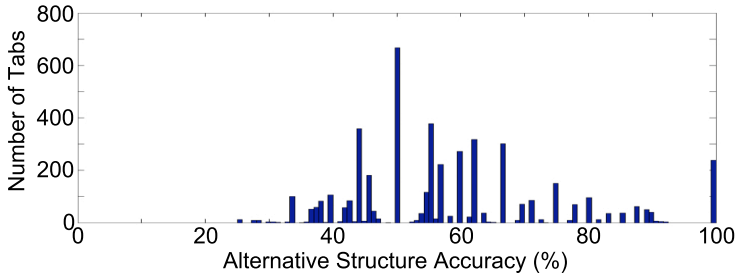


Figure 5.7: A histogram of the guitar tabs ASA.

CA	Domain	Tabs
66.52%	www.gtabs.org	180
66.50%	www.ttabs.com	1040
66.47%	www.tabondant.com	159
66.23%	www.guitarprotabs.org	167
64.88%	www.ultimate-guitar.com	1768
64.31%	www.bassmasta.net	160
63.89%	www.azchords.com	153
63.83%	www.guitaretab.com	373
63.22%	www.guitartabs.cc	1518
62.69%	www.e-chords.com	383

(a) Domains ranked by Chord Accuracy

SA	Domain	Tabs
53.31%	www.gtabs.org	155
52.00%	www.guitaretab.com	255
51.76%	www.guitarprotabs.org	121
51.64%	www.ttabs.com	915
51.48%	www.ultimate-guitar.com	1196
51.03%	www.guitartabs.cc	851
51.02%	www.e-chords.com	210
50.95%	www.tabsheaven.net	196
50.90%	flametune.com	121
49.52%	www.bassmasta.net	103

(b) Domains ranked by Structure Accuracy

Table 5.13: Average accuracy rates for different guitar tab domains.

5.6 Guitar Tab Toolkit

The Guitar Tab Toolkit[5] (GTT), written in Java, implements the guitar tab parsing system described in this chapter as well as other features. Users can use the crawling system to search for specific tabs using artist and song names, which returns URLs by default, *e.g.*

```
java -jar GTT.jar Led Zeppelin - Stairway To Heaven

Searching for tabs for Stairway To Heaven by Led Zeppelin
http://www.e-chords.com/tab.asp?idMusica=37160&tipo=T
http://www.ttabs.com/tabs.php?id=237913
http://tablatures.tk/view-text.php?id=24738
http://www.guitarprotabs.org/download.php?tabID=10097
http://www.tabs-database.com/led-zeppelin/Stairway-To-Heaven-Solo-tabs.html
...
```

[5]Available at www.eecs.qmul.ac.uk/∼robertm/gtt

Chord/structure sequences can be returned using the "-chords", "-structure" options respectively, *e.g.*

```
java -jar GTT.jar Coldplay - Yellow -chords

Searching for tabs for Yellow by Coldplay
Tab 1: B F# E B E G#m F#
Tab 2: C C/F G6 Fmaj9 F Am Gm7 G C C C Csus2 C G6 Fmaj9 C C G6 Fmaj9 C G6 Fmaj9
       C G6 Fmaj9 C Csus2 C F Am G F Am G F Am G Fmaj9 C G6 Fmaj9 C C G6 Fmaj9
       C G6 Fmaj9 C Csus2 C F Am G F Am G F Am G Fmaj9 C G6 Fmaj9 C C G6 Fmaj9
       C G6 Fmaj9 C Gm7 Fmaj9
Tab 3: B F# E E G#m F#
Tab 4: E A B G B Eb B Bsus4 B Bsus4 B F#6 Emaj7 B B F#6 Emaj7 B B F#6 Emaj7 B
       F#6 Emaj7 B F#6 Emaj7 B Bsus4 B Emaj7 G#m F#6 Emaj7 G#m F#6 Emaj7 G#m F#6
       Emaj7 G#m F#6 Emaj7 G#m F#6 Emaj7 G#m F#6 Emaj7 B F#6 Emaj7 B F#6 Emaj7 B
       Bsus4 B Emaj7 G#m F#6 Emaj7 G#m F#6 Emaj7 G#m F#6 Emaj7 B F#6 Emaj7 B F#6
       Emaj7 B F#m7 Emaj7
...
```

A specific tab can be examined if a URL is entered in place of an artist/track as the first argument, *e.g.*

```
java -jar GTT.jar url:
http://tabs.ultimate-guitar.com/j/jeff_buckley/hallelujah_ver2_crd.htm -chords

Tab 1: C Am C Am C Am C Am F G C G C F G Am F G Em Am F Am F C G C C Am C Am F
       G C G C F G Am F G Em Am F Am F C G C C Am C Am F G C G C F G Am F G Em
       Am F Am F C G C C Am C Am F G C G C F G Am F G Em Am C Am C Am F G C G
       C F G Am F G Em Am F Am F C G C
```

Finally, tabs specifically entered via their URL can be synthesised using "-synth" or saved to a MIDI file with "-midi" option, *e.g.*

```
java -jar GTT.jar
http://tabs.ultimate-guitar.com/g/guns_n_roses/sweet_child_o_mine_tab.htm
-midi /Home/Desktop/Tab.midi

Midi saved to /Home/Desktop/Tab.midi
```

5.6.1 Guitar Tab Synthesis

The synthesis function can be used to convert a tab into MIDI for audio playback or other purposes. Musical events are taken from the imported tablature notes and the chords (if not in a tab system line containing tablature) by using their notes described in the chord dictionary. Any tuning and/or capo declarations are taken into account when transforming these musical events into MIDI note numbers. An estimate of the timing of the MIDI event is made by the position of the note/chord along the line and as such the rhythm of the synthesised music is not precise.

5.6.2 Guitar Tab Score Following

The Guitar Tab Score Follower [Macrae and Dixon, 2010a] is a demonstration using the GTT and real-time DP algorithms from Chapter 3 to animate guitar tabs to follow a musicians progress through a song. This example application was created in Java with an interface designed in Max/MSP.[6] A screenshot of the application can be seen in Figure 5.8. Within the application the user has the option of loading a MIDI file, copying tablature text directly into the tab window or entering a tabs URL in the tab window. The controls allow the user to have the score animated, change the musical key of the piece, and select which DP method is used for synchronisation.

5.6.3 HOTTTABS

HOTTTABS[7] [Barthet et al., 2011] is an online multimedia guitar tuition service that gathers "hotttnesss" song popularity data from the Echo Nest,[8] guitar instruction videos from YouTube and guitar tabs crawled using the GTT to provide users with a selection of tabs for the most popular songs. The tabs are categorised according to their estimated difficulty based on their chord vocabulary (number of unique chords), as identified by the GTT. The links to the tabs are then presented to users in clusters showing their chord vocabulary as in Figure 5.9.

[6] http://cycling74.com/products/maxmspjitter/
[7] http://isophonics.net/hotttabs
[8] http://the.echonest.com/

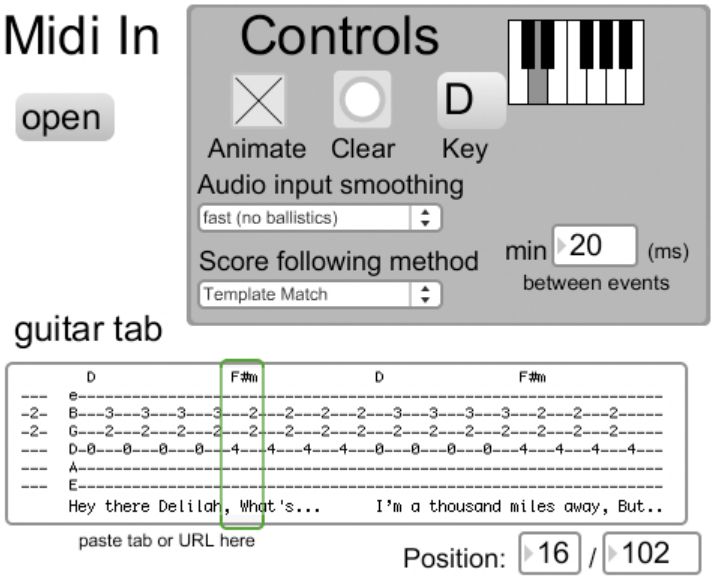


Figure 5.8: The Guitar Tab Score Follower.

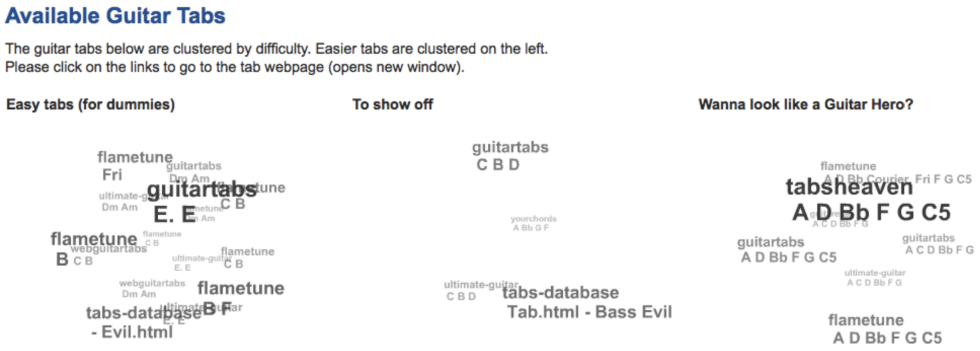


Figure 5.9: The HOTTTABS tab selection showing the clustering of tabs based on their chord vocabulary.

5.7 Conclusions

Online guitar tabs have been largely overlooked in MIR research despite being the most commonly available source of manual annotations on the internet. Reasons as to why tabs are not

used as much as they might be could be due to their lack of a standard, machine readable, format or the high probability that the tabs contain errors. In this chapter we have examined Guitar Tab Information Retrieval, the first attempt of its kind to automatically decode online guitar tabs and judge their accuracy regardless of their format. In order to achieve this, we have mined the internet for guitar tabs, created a heuristic based system for parsing tabs. Other data-mining approaches make use of machine learning for parsing such data and this approach would have been useful in this application and would have led to greater scalability. We used Dynamic Programming to measure the similarity of chords, chord sequences and tab structure, and packaged these functions together in the Guitar Tab Toolkit. The conventional method for calculating chord differences would be to use a substitution matrix based on the chords distance from each other on the circle of fifths, however DP was used in order to demonstrate the range of applications of DP.

In order to judge the accuracy of the tabs available online, we devised a test comparing tabs of The Beatles with known chord sequences and structure annotations. The results of this guitar tab evaluation have confirmed that online guitar tabs are often inaccurate, showing the average tab with chord content to have 63.1% accurate chords and the average tab with structure content to have 52.2% accurate structural indicators. However, we have also shown that there do exist a minority of highly accurate tabs. Were it possible to select the highly accurate tabs, it would be possible for musicologists to use the information held within to perform wide-scale analysis of chord patterns, song similarities and the evolution of playing styles/genres *etc.* As such, a tool for discriminating the correct tabs from the inaccurate, would not only make browsing tabs a more successful endeavour but also allow new avenues of MIR. Such a tool is the topic of the next chapter.

6

Filtering the Flood

"With the internet, everybody is a journalist now, everybody is a critic, everybody reviews albums" Noel Gallagher[1]

Contents

As we have shown in the previous chapter, the web holds vast collections of music annotations in the form of guitar tabs. Additionally, the web is used for sharing lyrics, music videos, standard sheet music notation, MIDI, and proprietary formats. Whilst some of these annotations are correct, the majority contain errors. Therefore, organising this data and sorting the correct from the defective, would be a great benefit for those musicians looking for the material. In this chapter, we will examine current ranking methods used on the web and how to judge those methods on their ability to prioritise accurate annotations in Section 6.1. Additionally in Section 6.1.6 we introduce the concept of "Concurrence" within data as a new ranking method. We evaluate the ranking methods with the guitar tabs gathered in the previous chapter in Section 6.2. In Section 6.3 we describe methods for mining the web for lyrics and evaluate ranking methods on these lyrics. We then conclude this work in Section 6.4.

[1] http://www.bbc.co.uk/news/entertainment-arts-14042533

6.1 Metadata Ranking Methods

In this Section we will describe how we will measure the effectiveness of ranking methods using correlation, how we will visualise this correlation, and describe three existing methods of ranking metadata on the internet.

6.1.1 Measuring Ranking Effectiveness With Correlation

In order to rank the ranking methods we require a measure of how well they rank metadata. For this purpose we use the correlation, first devised by Galton [1890], between the ranks attributed to the metadata and the accuracy of the metadata with respect to ground truth.

Two common correlations are described.

Pearson's Correlation Coefficient

The Pearson Product-Moment Correlation Coefficient [Pearson, 1895], or PCC, is widely used as a measure of the strength of the direct linear dependence between two variables. The following equation shows the PCC for variables X_i, Y_i:

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{6.1}$$

Spearman's Correlation Coefficient

One criticism of the PCC is that when a trend is curved, *i.e.* has a degree of polynomial greater than 1, the value of r is diminished. Spearman's Rank Correlation Coefficient [Spearman, 1904], or SCC, has been shown to detect general trends regardless of the degree of polynomial. As such, where ranks are concerned, the SCC may be a more appropriate measure as a ranks give discrete values without regard to the distance between elements. The following equation shows the SCC given the ranks x_i, y_i of variables X_i, Y_i.

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{6.2}$$

The PCC and SCC both return a value between -1 and 1 where higher negative values correspond to stronger negative correlations and higher positive values correspond to stronger positive correlations. A value of 0 indicates no correlation.

6.1.2 Visualising Correlation

A scatter plot is the simplest way to visualise correlation in data between two variables. When dealing with a large number of data points, the data can become obscure when points start to overlap on the graph and opacity is required in order to be able to see how crowded areas are. Unfortunately, the scatter graph function in MATLAB does not have an opacity variable. However, it is possible to export data to Photoshop from MATLAB. Therefore we have devised a MATLAB script for drawing scatter graphs in Photoshop with opacity: psscatter.m,[2] see Appendix B.

6.1.3 User Rating

Some webpages have a user ranking system which allow users to give feedback on how useful they think the data is, usually on a scale of 1 to 5. We refer to such data as the User Rating. It is expected that metadata with high user ratings would be more accurate than metadata with low ratings. User ratings are often employed in web usage mining. User ratings applied to music audio are often used in CF techniques [Borchers et al., 1998]. Slaney and White [2007] found playlists generated from artist similarities calculated from user ratings out-perform content based playlists.

6.1.4 Search Engine Results Page Rank

Search Engine Results Page (SERP) Rank is the order of results obtained from a search engine used to collect metadata. The most common example of this would be the Google SERP Rank. PageRank [Page et al., 1999] is an algorithm developed and used by Google to analyse the relevance of documents based on the hyper-links they contain (in addition to a set of over 200 other factors) and as such, using the SERP Rank is common in web structure mining. The SERP Rank value range is from 1 (the best) up until the rank of the least relevant document.

[2] Available at www.eecs.qmul.ac.uk/∼robertm/psscatter.m

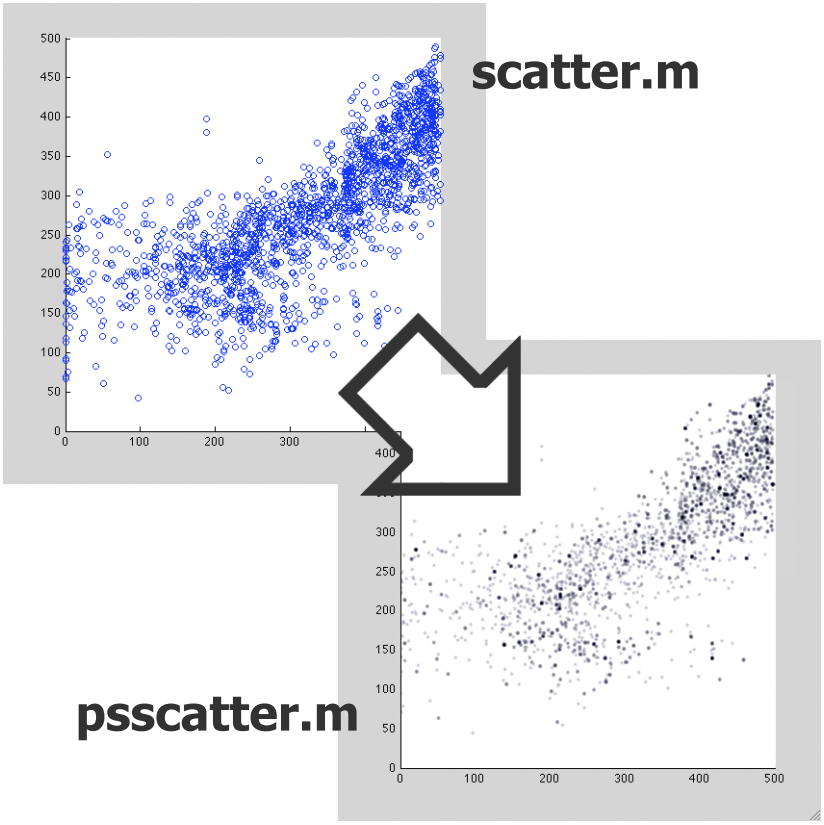


Figure 6.1: An example of loosely correlated data shown using the default scatter function in MATLAB (top left) and with the new psscatter function (bottom right).

6.1.5 Date Modified

If posted metadata is edited and reposted, it might be the case that more recent metadata is more accurate on average than earlier metadata. The Date Modified is used to determine how new the data is as it refers to the date in which a page was most recently updated. This value is expressed as the number of milliseconds since 00:00:00 January 1, 1970 GMT.

6.1.6 Concurrence

We propose a new method of ranking metadata according to their similarity with other metadata of the same subject, which we refer to as Concurrence (avoiding overuse of the word

similarity). The rationale behind Concurrence is that in a certain sense, the true metadata is what we, as people collectively decide it to be, as is evident by the collection of ground truth data typically involving taking the average of a number of users' input. Therefore our hypothesis is that metadata which closely resembles other instances is more likely to be accurate than independent outliers that have a smaller correspondence with the rest of the data. We measure this Concurrence factor by averaging the similarity of metadata to its alternatives. The Concurrence factor can be viewed as a Multiple Classifier System [Marti and Bunke, 2001; Ho et al., 1994] similar to a Vote Fusion or Sum Fusion [Kittler and Alkoot, 2003].

6.2 Ranking Guitar Tabs

In this Section we evaluate the ranking methods for guitar tabs using the same 24746 tabs we mined in Chapter 5. We first adapt the ranking methods outlined previously and then evaluate these methods by measuring their correlation with the tabs' CA, SCA, SA, and ASA, as described in Sections 5.4.5, 5.4.6, 5.4.8, & 5.4.9.

6.2.1 Test Data: The Beatles

The ground truth in this experiment is the same chord sequence and structure segmentation data for 180 tracks by The Beatles as described in 5.1.

6.2.2 Tab Ranking Methods

The following methods describe how we apply the ranking methods to the guitar tab data.

User Rating

The User Rating is the average user rating assigned to the tab at www.911tabs.com from 1 (bad) to 5 (good). The number of votes that went into this average rating is not provided by the tab site. 1246 tabs with identifiable chord content had User Ratings.

SERP Rank

The tab's SERP Rank corresponds to where the URL of the tab is found in the ordered list of Google's ranked search results [Page et al., 1999]. Values range from 1 (best) to 100 (worst

known), as our mining was restricted to the top 100 results (Section 5.2.1). 5619 of the tabs found had SERP Ranks associated with them, 1931 of which had identifiable chord sequences.

Date Modified

2022 of the tabs with chord sequences had an associated last date modified that was greater than 0. Any value of 0 is ignored as it was presumed to be more likely that such a date was unknown, as opposed to the tab being written for ARPANET.

Chord Concurrence (CC)

To determine how well chord sequences of songs agree with each other, we define the Chord Concurrence as the average of the similarities between a tab's chord sequence T_k and the chord sequences $T_i (i \neq k)$ of all the other tabs of the same song.

$$\mathrm{CC}(T_k) = \sum_{i=1, i \neq k}^{n} \mathrm{CSS}(T_k, T_i)/(n-1) \tag{6.3}$$

Structure Concurrence (SC)

The Structure Concurrence is the average of the similarities between a tab's structural sequence T_k and the structural sequences T_i of all the other tabs of the same song.

$$\mathrm{SC}(T_k) = \sum_{i=1, i \neq k}^{n} \mathrm{SS}(T_k, T_i)/(n-1) \tag{6.4}$$

Tab Length

The Tab Length is the number of tablature lines in the tab.

Chord Sequence Length

The Chord Sequence Length is the number of chords in the tabs chord sequence.

Structure Length

The Structure Length is the number of structural sections in the tab.

6.2.3 Evaluation

Table 6.1 shows the correlations found between the tabs' CA, SCA and 4 relevant features discussed above. Similarly, we give the correlations with the SA and ASA in Table 6.2. For the sample sizes provided; the required absolute value for statistical significance is approximately 0.1. Surprisingly, the rating given by users and the date the tab was modified had no statistically significant correlation with the accuracy of the tabs' chord or structure sequence in any of the measures used. The SERP Rank did show a statistically significant correlation when using the SCC for chord accuracy measures and also showed statistically significant correlations for the structural accuracies of the tabs. The SERP Rank is expected to have a negative correlation due to the reverse ordering of the page ranks (1 being best, 100 being worst). A strong correlation was provided by the Concurrence methods that had a PCC of 0.541 for CA, 0.517 for SCA, 0.283 for SA and 0.165 for ASA. For the Chord Accuracy and Segment Chord Accuracy correlations, the Tab Length and Chord Sequence Length showed statistically relevant correlations greater than the typical ranking methods. Additionally, Tab Length and Structure Length shows statistically significant correlations with the Alternative Structure Accuracy measurement.

These results show it is possible to improve the ranking of tabs by search engines based on analysing the contents of the metadata. Simply looking at the amount of tab content improves the ranking methods but the best methods compare the metadata with others of the same track. Spearman's Rank Correlation Coefficient (SCC) was used alongside the more common Pearson Product-Moment Correlation Coefficient in this test to ensure the results weren't biased by the degree of separation within the ranking values. With the exception of the Date Modified, the SCC had a positive effect on the correlation results of all the ranking methods with CA, most notably in the case of SERP Rank which was statistically relevant only using the SCC measurement of correlation.

Six of the correlations from Tables 6.1 & 6.2 are represented in scatter plots in Figure 6.2. Each point represents the CA or SA (vertical coordinate) plotted against a ranking method's value (horizontal coordinate) for a single tab. The ranking methods shown are the User Rating (Figures 6.2a & 6.2b) SERP Rank (Figures 6.2c & 6.2d), and Concurrence (Figures 6.2e & 6.2f). A negative correlation is apparent in the scatter plots of Figures 6.2c & 6.2d, show that

Ranking Method	PCC (r)		SCC (ρ)		Samples
	CA	SCA	CA	SCA	
Chord Concurrence	0.541	0.517	0.548	0.506	7541
User Rating	0.067	0.061	0.093	0.089	1159
SERP Rank	-0.074	-0.077	-0.124	-0.128	1931
Date Modified	0.052	0.026	0.029	0.012	1666
Tab Length	0.179	0.132	0.231	0.194	7541
Chord Sequence Length	0.182	0.306	0.310	0.308	7541

Table 6.1: Correlations between various ranking methods and the Chord Accuracy (CA) and Segment Chord Accuracy (SCA).

Ranking Method	PCC (r)		SCC (ρ)		Samples
	SA	ASA	SA	ASA	
Structure Concurrence	0.283	0.165	0.241	0.137	4643
User Rating	0.085	0.081	0.050	0.048	620
SERP Rank	-0.114	-0.136	-0.117	-0.120	1197
Date Modified	0.052	0.072	0.051	0.070	1283
Tab Length	0.002	-0.009	0.109	0.138	4643
Structure Seq Length	0.013	0.057	0.101	0.167	4643

Table 6.2: Number of samples and correlation values between various ranking methods and the Structure Accuracy (SA) and Alternative Structure Accuracy (ASA).

tabs with lower SERP Ranks are more accurate. A stronger trend can be seen in Figures 6.2e & 6.2f, where the tabs with a higher Chord/Structure Concurrence have are more accurate.

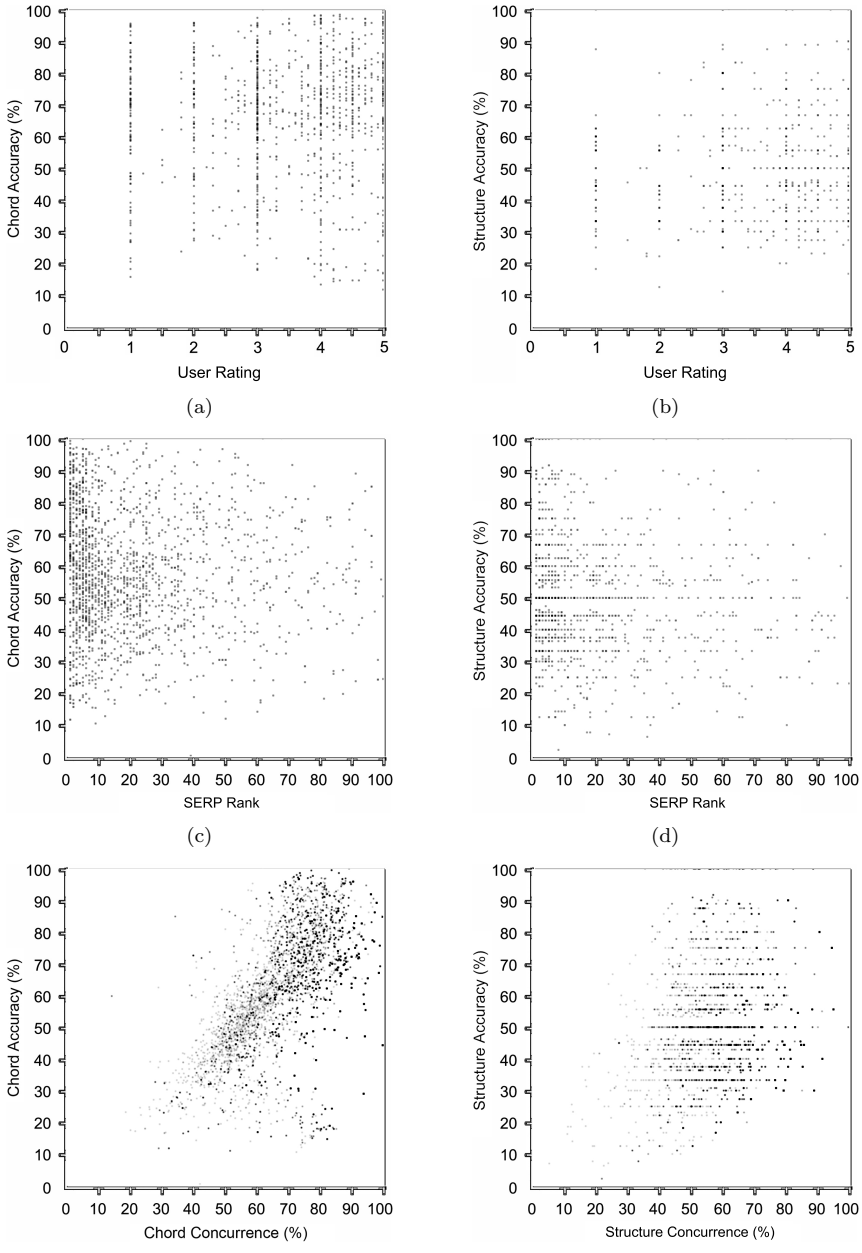


Figure 6.2: Scatter graphs showing the trends between CA and SA for three ranking methods.

6.2.4 Is Chord Concurrence Biased Due to Sharing CSS With CA?

One criticism of the CC measurement is that it is based on the same CSS measurement that the CA metric uses. We therefore propose an alternative CC method based on a CSS featuring LCSSeq. Unlike DTW, the LCSSeq returns a discrete value depending on the length of the longest common sub-sequence. We convert this to a distance score and express it as a percentage:

Chord Sequence Similarity Using LCSSeq(CSS-LCSSeq)

$$\text{CSS-LCSSeq}(T_1,T_2) = \left(\frac{\text{LCSSeq}(T_1, T_2, CS)}{\max(|T_1|, |T_2|)}\right) \times 100 \tag{6.5}$$

Chord Concurrence Using LCSSeq (CC-LCSSeq)

$$\text{CC-LCSSeq}(T_k) = \sum_{i=1, i\neq k}^{n} \text{CSS-LCSSeq}(T_k, T_i)/(n-1) \tag{6.6}$$

Repeating the experiments from before we find the correlation between CC-LCSSeq and CA is similar to that of the original CC, as shown in Table 6.3. There is a slightly weaker correlation but this may be due to the greater degree of accuracy offered by the DTW measurement which uses the Chord Similarity metric as opposed to a binary cost function.

Ranking Method	PCC (r)		SCC (ρ)		Samples
	CA	SCA	CA	SCA	
CC	0.541	0.517	0.548	0.506	7541
CC-LCSSeq	0.464	0.456	0.498	0.471	7541

Table 6.3: An extension of the results in Table 6.1 comparing the alternative Chord Concurrence measures.

6.2.5 Is Chord Concurrence Dependent on Sample Size?

A potential weakness of the Concurrence methods could be in being dependent on the number of tabs available for a particular song. To see if this would affect performance, we calculated the correlation between N (the number of tabs for a particular song) and C (correlation between CA and CC) for each of our 180 tracks. The result, -0.030, is not statistically significant for the

sample size of 180 songs, suggesting that Chord Concurrence is a relevant indicator of Chord Accuracy regardless of the number of tabs on which it is based.

6.2.6 Chord Detection With Guitar Tabs

In addition to evaluating the ranking methods on their ability to rank the guitar tabs, we also look at how they can be used for chord detection and compare these methods with automatic chord recognition. Using online chord sequences in chord detection tasks has been previously studied by McVicar et al. [2011a,a] who used chord sequences to assist automatic machine listening chord detection systems. Ranking methods and automatic chord recognition systems satisfy the same information need: finding the chords to a given song. We selected the top ranking tab for each feature and compared the accuracy of its chord sequence with the output of a state-of-the-art automatic chord detection system [Mauch and Dixon, 2010]. We refer to this automatic method as "Auto (Mauch)". Additionally, for comparison purposes, we select the chord sequence from the tabs that has the highest Chord Sequence Similarity 5.4.4 with the chord sequence produced by the automatic chord detection system and refer to this tab selection method as "Auto-Similar Tab". We also select the tab which is most similar to the ground truth, labelled the "Most Accurate Tab" to represent the best possible accuracy achievable from the guitar tabs.

In Table 6.4 there is an example of the chord sequences produced by the automatic chord recognition system, those selected by our features, and the ground truth annotations for The Beatles' "Don't Pass Me By". The chord accuracies are also given. Table 6.5 shows the average accuracy of the methods. There is a clear superiority in the automatic detection algorithm which is 10% more accurate, on average, than the tabs selected by any of our features. Of the features, the Chord Concurrence is the most successful feature for selecting the tab to use.

Source	CA	Chord Sequence
Ground Truth	-	C F G F C F G F C F C G F C F G F C F C G F C C5 C F C G F C F G Csus4/A C
Auto-Similar Tab	97.00%	C F G F C F G F C F C G F C F G F C F C G F C F C G F C F G D C
Auto (Mauch)	88.96%	C F G F C5 F G F C Cm7 F C G F C F G F C5 F C G F C G G7 C Cm C F C G F C F G C C5 F
Chord Concurrence	87.75%	C F G F C F G F C F C G F C F G F C F C G F C
User Rating	81.86%	C F G F C F G F C F C G F C F G F C
SERP Rank	61.52%	G C D C G C D C G C G D C G C D C G C G D C G C G D C G
Date Modified	61.52%	G C D C G C D C G C G D C G C D C G C G D C G C G D C G

Table 6.4: Example chord sequences retrieved by the various chord detection methods for the song "Don't Pass Me By" showing the Chord Accuracy (CA) of these sequences.

Detection Method	Chord Accuracy
Most Accurate Tab	84.48%
Auto-Similar Tab	**81.27%**
Auto (Mauch)	**79.12%**
Chord Concurrence	71.49%
User Rating	69.89%
SERP Rank	68.26%
Date Modified	65.85%
Randomly Selected	63.14%

Table 6.5: The average Chord Accuracy of the chord sequences, from over 180 Beatles tracks, that were provided by the top-ranked tabs and the chord detection methods. The first row shows the average of the most accurate tabs and therefore represents the highest possible score for a tab selection based chord detection method. The Auto-Similar Tab and Auto (Mauch) methods in rows two and three make use of the audio and Auto (Mauch) is the only method to not select an online annotation. The final row shows the average as if the tab was randomly selected.

6.3 Ranking Lyrics

In this section we are concerned with ranking web based song lyrics. Unlike lyric selection/retrieval methods [Knees et al., 2005; Korst and Geleijnse, 2006], we are simply concerned with rating existing lyrics, so that users can apply their own selection should the first result not be appropriate. To the best of our knowledge, lyrics ranking has only previously been attempted as part of more generalised web resource ranking methods [Page et al., 1999]. In order to evaluate song lyrics ranking we first describe a test data set for this purpose and how we then proceed to mine the web for lyrics of the songs in this dataset. We then formulate a metric to compare each lyric to the ground truth, as an accuracy measurement, and to other versions to calculate the Lyric Concurrence. We then adapt the ranking methods outlined previously to evaluate these methods by measuring their correlation with the lyrics' accuracy.

6.3.1 Test Data: The musiXmatch Dataset

The Million Song Dataset (MSD)[3] is a collection of metadata for a million popular music tracks provided by Bertin-Mahieux et al. [2011] in collaboration with The Echo Nest. A subset of this, called the musicXmatch Dataset (MXMD),[4] consists of 237,662 lyrics to songs within the MSD provided in a Bag-of-words format with the 5000 most common (stemmed) words.

Bag-of-words Format

The Bag-of-words format (BOW) is primarily a means of summarising text by listing the unique words with the number of occurrences of each word in the text, with all punctuation removed. These word and count pairs are ordered by their count with the most common coming first. For example:

"On mules we find two legs behind and two we find before. We stand behind before we find what those behind be for."

can be represented in BOW format as:

"we:4, find:3, behind:3, two:2, before:2, on:1, mules:1, legs:1, and:1, stand:1, what:1, those:1, be:1, for:1"

[3] http://labrosa.ee.columbia.edu/millionsong/
[4] http://labrosa.ee.columbia.edu/millionsong/musixmatch

Additionally the words are stemmed [Lovins, 1968] so that words with different endings are reduced to their root form, reducing the number of unique words. Using this BOW format avoids copyright issues with sharing lyrics for the purposes of research.

6.3.2 Lyrics Mining

For each of the 237,662 tracks tracks in the MXMD we searched DogPile[5] for lyrics using the following terms "<artist><song>lyrics -video". Previous web mining approaches have used the Google Web API in a similar fashion [Korst and Geleijnse, 2006; Knees et al., 2005], however we required a search engine with an unrestricted number of searches. From the list of URLs returned by this search we selected only those that contained the song title in the URL. This set of URLs provides a similar representation of the URLs a user might select when manually searching for lyrics. 888,745 URLs were found using this method for the 237,662 tracks. In order to extract the lyrics from the URLs we separated and analysed each line to determine whether it contained lyrics-like text and then selected the longest sequence of lyrics-like text lines in the page. Any lyrics that were less than three lines or over 200 lines long were discarded. As we are interested in comparing with Concurrence, we discarded songs and their lyrics if they had less than three lyrics associated with the song. The lyrics extraction process is demonstrated in Figure 6.3.

6.3.3 Lyric Accuracy (LA)

In order to calculate the accuracy of the lyrics we first convert the lyrics to the BOW format with the 5000 most common stemmed words using the same stemming code the MXMD used. We describe the ground truth MXMD BOW $G = (g_1, g_2, ..., g_M)$ and the lyrics BOW $L = (l_1, l_2, ..., l_N)$ as sets of word (w_i) and count (x_i) pairs where $g_i = (w_i, x_i)$. Each word in the ground truth BOW G is looked for in the lyrics BOW L so that if a match is found *i.e.* $g_m(w) = l_n(w)$. Therefore each ground truth word yields an expected word count $g_m(x)$ and a found word count of $l_k(x)$ if the word was present in the lyrics BOW and 0 if not. If the found word count is greater than the expected word count, the found count is replaced as the expected count minus the difference or 0 if this difference is greater than the expected count.

[5] http://www.dogpile.com/

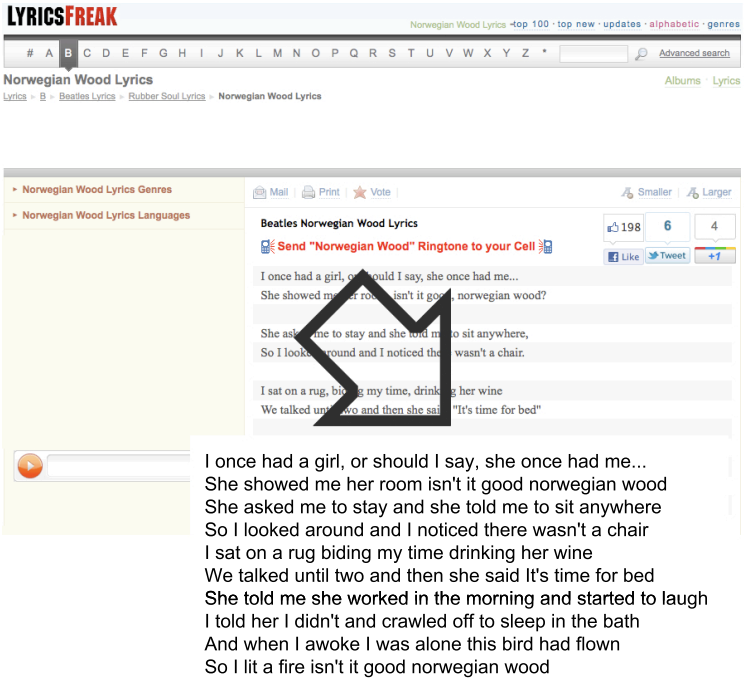


Figure 6.3: An example lyrics web page and the lyrics extracted from it.

The LA is calculated as the sum of the found word counts divided by the sum of the expected word counts multiplied by 100 and divided by the sum of the ground truth counts *expected*, so as to be expressed as a percentage. Equation 6.7 shows this calculation and Table 6.6 shows an example of the LA measurement.

$$\mathrm{LA}(G,L) = \frac{\sum \max(g_m(x) - |g_m(x) - l_k(x)|, 0)}{\sum g_m(x)} \times 100 \tag{6.7}$$

Ground Truth: “Are we human or are we dancer? My sign is vital, my hands are cold”
Lyrics: “Are we human or are we dancers? My signs are vital, my hands are cold”
Lyrics Accuracy (LA): $(12/14) \times 100$ 85.7%
(wrong count for “is” and wrong count for “are”)

Table 6.6: Lyrics Accuracy (LA) example.

6.3.4 Lyrics Similarity (LS)

The Lyrics Similarity is a measure of how similar two lyrics, L_1 and L_2 are. We use the LED of the entire sequence of characters in both lyrics, not stemmed and with all the punctuation included. We convert the LED to a similarity score:

$$\mathrm{LS}(L_1, L_2) = \left(1 - \frac{\mathrm{LED}(L_1, L_2)}{\max(L_1, L_2)}\right) \times 100 \tag{6.8}$$

For the Lyrics Ranking experiments we additionally tried a variation of the LS called LS_{ns} where spaces are removed from the input lyrics L_1 and L_2. The incentive for removing spaces is that, as the average english word length is 5 characters, spaces make up roughly $\frac{1}{6}$ of the text and possibly contain less relevant information than the rest of the text. As the LED has quadratic costs, reducing the input sequences by $\frac{1}{6}$ reduces the processing time and memory requirements by 31%.

Lyrics 1: "On mules we find two legs behind and two we find before."
Lyrics 2: "We stand behind before we find what those behind be for."
Lyrics Similarity (LS): 43.8%
Lyrics Similarity no spaces (LS_{ns}): 45.7%
Lyrics 1: "Are we human or are we dancer? My sign is vital, my hands are cold' '
Lyrics 2: "Are we human or are we dancers? My signs are vital, my hands are cold"
Lyrics Similarity (LS): 92.9%
Lyrics Similarity no spaces (LS_{ns}): 90.9%
Lyrics 1: "Scaramouche, Scaramouche, will you do the Fandango"
Lyrics 2: "Scallaboosh, Scallaboosh, will you to the banned tango"
Lyrics Similarity (LS): 69.1%
Lyrics Similarity no spaces (LS_{ns}): 66.0%
Lyrics 1: Radiohead - High and Dry (azlyrics.com/lyrics/radiohead/highdry.html)
Lyrics 2: Jamie Cullum - High and Dry (azlyrics.com/lyrics/jamiecullum/highanddry.html)
Lyrics Similarity (LS): 86.6%
Lyrics Similarity no spaces (LS_{ns}): 86.0%

Table 6.7: Lyrics Similarity (LS) examples.

6.3.5 Lyrics Statistics

The final list of lyrics included 358535 lyrics for 67156 songs with an average Lyrics Accuracy of 38.6%. The distribution of the lyrics over these songs can be seen in Figure 6.4. This lyrics distribution shows a quick drop off in the number of lyrics per song after the songs with less than three lyrics were removed. The range of lyrics accuracy results can be seen in the histogram in Figure 6.5. With the exception of a spike in very innacurate (possibly non-lyrics) results, this graph resembles the range of guitar tab chord accuracies (Figures 5.4 and 5.5) and guitar tab structure accuracies (Figures 5.6 and 5.7) seen earlier. The large number of low accuracy lyrics and the low average Lyrics Accuracy suggest the lyrics mining procedure failed to filter out all the non-text lyrics, however, this is not a trivial task for users browsing the web either and so we consider these non lyrics within the ranking method experiments as one purpose of these is to differentiate between lyrics and non-lyrics. In Section 6.3.8 we examine the possibility of removing these non-lyrics to judge their effect on the ranking experiments. Table 6.8 shows the top twenty lyrics domains based on their average Lyrics Accuracy. The increase in Lyrics Accuracy of these domains over the average suggest that a simple filter restricting the results to known lyrics domains would remove most of the non-lyrics.

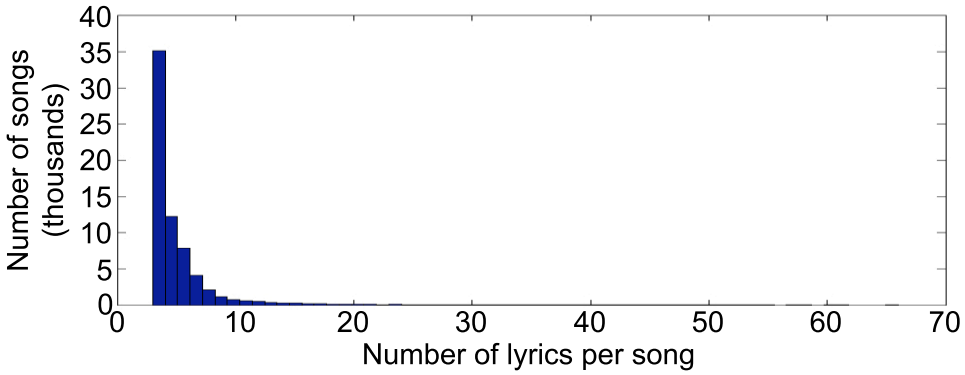


Figure 6.4: A histogram showing the distribution of lyrics for the 61755 songs.

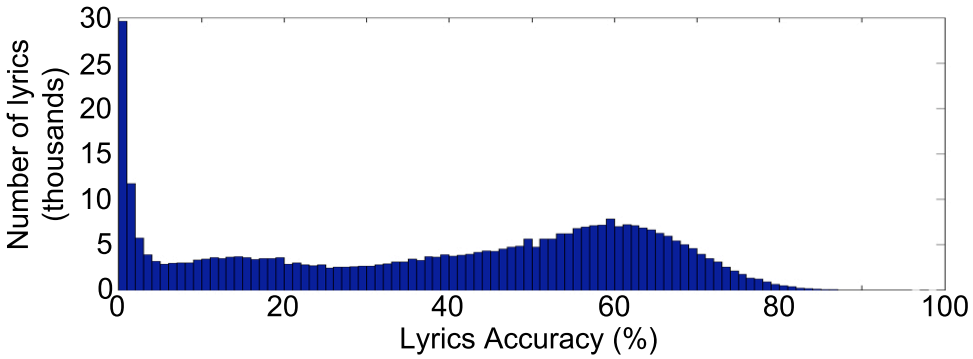


Figure 6.5: A histogram showing the distribution of the lyrics accuracies.

LA	Domain	Lyrics
55.82%	www.alivelyrics.com	123
52.75%	www.sing365.com	15798
52.53%	www.popular-lyrics.com	142
52.43%	www.plyrics.com	127
52.34%	www.musicsonglyrics.com	3307
52.33%	www.lyricspond.com	535
52.25%	www.songteksten.nl	1178
51.97%	www.lyricsdepot.com	3301
51.93%	www.azlyrics.com	7006
51.30%	www.1songlyrics.com	253
51.11%	www.absolutelyrics.com	1360
51.02%	www.lyricsondemand.com	2909
50.85%	www.sarkisozum.gen.tr	138
50.72%	www.christian-lyrics.net	167
50.62%	www.lyricsdomain.com	925
50.57%	www.lyricstop.com	235
50.14%	www.cowboylyrics.com	1656
49.26%	www.lyriczz.com	682
49.08%	www.lyricsreg.com	1877
49.01%	www.lyricmania.com	155

Table 6.8: Average accuracy rates for different lyrics domains.

6.3.6 Lyrics Ranking Methods

The following methods describe how we apply the ranking methods to the lyrics.

SERP Rank (SE)

The lyric's SERP Rank corresponds to where the URL of the lyric is found in the ordered list of DogPile's ranked search results. Values range from 1 (best) to 100 (worst known), as our mining was restricted to the top 100 results (see Section 6.3.2). All the lyrics were mined using DogPile and as such had an associated SERP Rank.

Date Modified (DM)

137875 of the 358535 lyrics had an associated last date modified that was greater than 0. Any value of 0 is ignored as it was presumed that such a date was unknown.

Lyrics Concurrence (LC)

To determine the extent to which lyrics of songs agree with a set of lyrics, we measure the Lyrics Concurrence as the average of the Lyrics Similarities between a lyric L_k and the other lyrics of the same song $L_i (i \neq k)$.

$$\mathrm{LC}(L_k) = \sum_{i=1, i \neq k}^{n} \mathrm{LS}(L_k, L_i)/(n-1) \tag{6.9}$$

Lyrics Concurrence No Spaces ($\mathbf{LC}_{ns}$)

Additionally, we measure the Lyrics Concurrence No Spaces as the average of the LS_{ns} between a lyrics' L_k and the other Lyrics of the same song $L_i (i \neq k)$.

$$\mathrm{LC}_{ns}(L_k) = \sum_{i=1, i \neq k}^{n} \mathrm{LS}_{ns}(L_k, L_i)/(n-1) \tag{6.10}$$

6.3.7 Evaluation

Table 6.9 shows the correlations found between the lyrics LA and the 4 ranking methods described above. Remarkably, the correlations match very closely what we found with the guitar

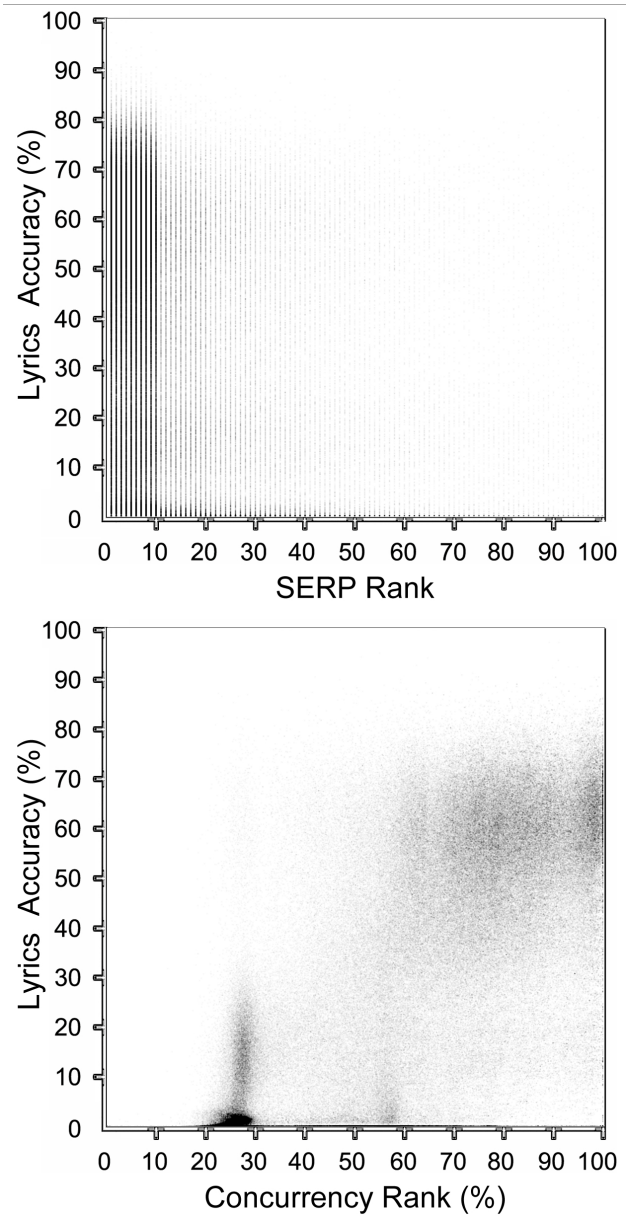


Figure 6.6: Scatter graphs showing the trends between LA and respectively the SERP Rank (above) and Lyrics Concurrence (below) on 358535 lyrics.

tabs with concurrence having the strongest correlation, the SERP Rank having a weak correlation and the Date Modified having a very low correlation. Comparing LC and LC_{ns} we find that discarding spaces improves the correlation slightly therefore LC_{ns} improves performance in both accuracy and efficiency. The results of this experiment show once again that analysing the content of the metadata in comparison to the other metadata available leads to a better ranking system than methods based on user statistics and link analysis or the date modified.

Ranking Method	PCC (r)	SCC (ρ)	Samples
	LA		
Lyrics Concurrence	0.654	0.607	358535
Lyrics Concurrence No Spaces	0.657	0.609	358535
SERP Rank	-0.206	-0.190	358535
Date Modified	0.016	0.012	137875

Table 6.9: Number of samples and correlation values between various ranking methods and the Lyrics Accuracy (LA).

6.3.8 To What Extent do Non-Lyrics Affect Ranking Correlations?

As mentioned previously, the Lyrics data contains many files that are not lyrics at all and this may affect the correlations. We therefore repeat the ranking methods experiment excluding the files that have a Lyrics Accuracy of less than 10%, the results of which are shown in Table 6.10. The ranking methods all see a reduction in the correlation between rank and Lyrics Accuracy. This difference suggests that the methods help distinguish the lyrics from the non-lyrics.

Ranking Method	PCC (r)	SCC (ρ)	Samples
	LA		
Lyrics Concurrence	0.477	0.477	289346
Lyrics Concurrence No Spaces	0.484	0.484	289346
SERP Rank	-0.191	-0.191	289346
Date Modified	0.009	0.033	107661

Table 6.10: A modified version of Table 6.9 showing correlation values between various ranking methods and the Lyrics Accuracy (LA) without the lyrics with an LA of less than 10%.

6.3.9 Is Lyrics Concurrence Dependent on Sample Size?

To see if the number of lyrics available for a particular song effects the correlation of Lyrics Concurrence with lyrics Accuracy, we calculate the correlation between N (the number of lyrics for a particular song) and C (correlation between LA and LC) for each of the 61755 songs. The result, 0.074, is not statistically significant for the sample size, suggesting that Lyrics Concurrence, like Chord Concurrence is a relevant indicator of accuracy regardless of the number of lyrics on which it is based.

6.3.10 Song Lyrics Detection

As with guitar tabs, we try the ranking methods as lyrics detection systems by taking the highest ranking lyrics for each of the 61755 songs. Table 6.11 shows the average accuracy of the ranking methods. Of the ranking methods, the Lyrics Concurrence is the most successful feature for selecting the most accurate lyrics to use.

Detection Method	Lyrics Accuracy
Lyrics Concurrence	47.3%
Date Modified	43.6%
SERP Rank	42.5%
Randomly Selected	38.6%

Table 6.11: The average Lyrics Accuracy of the top ranked lyrics over 61755 tracks (41614 tracks in the case of Date Modified as 38.5% of the lyrics don't have an associated date). The final row shows the average as if the lyrics were randomly selected.

6.4 Conclusions

In this chapter we have examined how well metadata ranking methods correlate with accuracy using two examples, guitar tabs and lyrics. In addition we have examined potential weaknesses of the ranking algorithms described (Sections 6.2.5, 6.3.8, and 6.3.9) and found that the metadata concurrence methods' correlations with accuracy are independent of the sample size or ranking quantisation. Another question one might ask about the legitimacy of the tab ranking tests is that the concurrence method uses the same similarity measurement (Section 5.4.4 &

5.4.7) as the accuracy measurement. However, this concern is negated by the second experiment with lyrics that shows a stronger correlation despite the methods used for calculating the Lyrics Accuracy and the Lyrics Concurrence being different. The rationale of the Concurrence factor is that the correctness of metadata, is determined by agreement of expert human annotators. This is how ground truth is usually authored when evaluating machine listening related tasks within MIR. The fact that human annotators of chord sequences have failed to match the performance of the machine listening chord detection system in Section 6.2.6 shows the importance of the expertise in the human annotators and how automatically produced data might be more reliable ground truth data in certain instances. However, it must be pointed out that the human annotators in this case may not have been attempting the same task as the chord detection method in that the annotations they produce may be for accompaniment or to play the song in an alternative style to the original. The Concurrence factor was found to be the most reliable indicator of metadata accuracy and this may be due to being the only web page-ranking method to analyse the content of the metadata, unlike the other methods which are based primarily on users' actions. Additionally it must be noted that the SERP Rank and User Rating are not primarily aimed at showing the most accurate metadata first, but rather the most preferred as judged by the viewers who give feedback in ratings and views.

7 Conclusions

"There's no problem, only solutions." John Lennon

Contents

In this thesis we have examined the challenges concerning linking music and metadata. We have developed linking methods, conducted evaluations and filtered quality metadata from the data deluge that is the web. We have answered each of the research questions proposed in Chapter 1. In order to improve the efficiency of DP at synchronising music and metadata without significant loss in accuracy, we developed five real-time and linear modifications of DTW in Chapter 3. We then evaluated these methods and other real-time modifications of DTW in Chapter 4 as an example of how we can judge the capabilities of music metadata synchronisation methods. In Chapter 5 we showed that text-mining techniques can be used to decipher the musical annotations within online guitar tabs and we assessed their accuracy using DP methods. In Chapter 6 we then applied DP and the text-mining techniques to filter accurate guitar tabs and lyrics from the web.

In this chapter, we will summarise the research contributions made in this thesis in Section 7.1 and other contributions in Section 7.2. We will then raise some discussion points that came

about during this work in Section 7.3. Finally, in Section 7.4 we will outline future research questions that have been opened up by this research.

7.1 Summary of Research Contributions

The following is a list of the main research contributions of this thesis.

New Metadata Alignment Methods

In Chapter 3 we presented DTW modifications for improving the efficiency of music audio and metadata alignment, including: the Greedy method (Section 3.2), The Two-Step algorithm (Section 3.3), Windowed Time Warping (Section 3.4), On-Line Time Warping Jumping (Section 3.5) and On-Line Time Warping Constrained (3.6). Additionally we presented a method for finding alternative alignments that do not necessarily start at the beginning of sequences in The Quick Start Method (Section 3.7).

Evaluation of Real-Time DTW based methods

Within the MetaSync application, we ran an evaluation of real-time DTW methods using MIREX data and the Mazurka dataset. This evaluation included the five real-time Dynamic Programming modifications proposed in Chapter 3 along with two variations of OTW and compared the results to MIREX results based on the same dataset. Additionally the computational efficiency of the methods was examined and compared with that of FastDTW. We achieved comparable results with leading probabilistic methods in score following trials, with WTW reaching 89.0% of notes accurately aligned within 2000ms. We also achieved linear computational and memory costs in aligning sequences up to 1 million frames long with what typically amounts to 5 and a half hours of audio being aligned in less than two minutes for all the five proposed methods. This evaluation found that real-time Dynamic Programming methods can be as accurate as state-of-the-art score following applications and computationally efficient as to scale to large datasets.

Analysed Guitar Tabs and Lyrics

In Chapters 5 and 6 we wrote data-mining functions for extracting metadata from guitar tabs and lyrics and analysed their accuracy using custom metrics. We found that tabs had an average chord accuracy of 63.1% and that duplicates were on average slightly more accurate. We also found lyrics were on average 38.6% accurate due to non-lyrics (noise) being hard to avoid.

The Concurrence Factor

The Concurrence Factor, introduced in Chapter 6, is a method for ranking music metadata based on the similarity of its musical content to other versions of the same song. We have shown that Concurrence is a reliable indicator of accuracy with three types of annotations (chord sequence, structure sequence and lyrics), consistently providing a greater correlation with the accuracy of the metadata than the date modified, user rating or SERP Rank.

7.2 Summary of Other Outputs

This section is a summary of the additional outputs of the work in this thesis.

An Application for Aligning Music and Scores

MetaSync, presented in Chapter 3, is a music and score alignment application that implements all of the Dynamic Programming modifications in Chapter 3 as well as standard DTW. MetaSync accepts audio and MIDI as inputs and can output MIDI, text or Sonic Visualiser session files. MetaSync makes automatically aligning music audio and scores for the purpose of producing ground truth automated datasets easier.

The Mazurka Dataset

Using the MetaSync application, we created a new score following Mazurka dataset in Chapter 4. This dataset was adapted from reverse conducting data by the CHARM Mazurka Project [Sapp, 2007].

The Guitar Tab Toolkit

The GTT incorporates the text mining techniques we used in Chapter 5 to analyse the musical annotations in online tabs. Using GTT, users can mine the web for guitar tabs of a particular song and extract the chords, structure or MIDI from the tabs found.

7.3 Discussion Points

This work in thesis has contributed to the following topics within MIR.

Sample Size

Although we now have access to unprecedented quantities of music and metadata, rarely in MIR have we taken advantage of them. Due to issues in licensing, academics in MIR have consistently used small scale tests that do not show if methods are scalable to real services [Bertin-Mahieux et al., 2011]. Since the first Music Information Retrieval Evaluation eXchange in 2005 [Downie et al., 2005] the dataset collections for the tasks have rarely changed and there is a recognised need for them to be much larger [Urbano, 2011]. Music service providers and applications for interacting with music require music libraries that contain every song a user might wish to listen to. The scale of such data requires any musical algorithms to run with the absolute maximum efficiency and linear computation is rarely feasible. One issue of scalability is in precision and handling false positives and the concern is that methods that work for 20 songs won't necessarily work for 2 million. Cover songs, alternative recordings, re-masterings, re-mixes and changing band line ups can confuse music identification tasks as well as our sense of what is a song. Another issue with small datasets is that they are less likely to comprehensively represent all the genres and specific styles that make up online music libraries. For this reason, MIR needs to make use of the APIs, web data and cloud services to increase the scale of method testing. The guitar tab, lyrics, and ranking evaluations presented in Chapters 5 and 6 use large datasets to ensure the data best represents the music metadata available online. Additionally, two new datasets were announced at the 12th International Society for Music Information Retrieval Conference (ISMIR 2011) that set new precedents of scale in MIR. The Million Song Dataset [Bertin-Mahieux et al., 2011] provides a collection of audio features from various APIs and Burgoyne et al. [2011] curated a set of over 1000 expertly annotated ground

truth chord transcriptions. It is hoped that datasets like these will allow MIR research to focus on scalable methods that are applicable to real world services. Instead of worrying about fair representation and over-training, we can use samples that encompass such a large quantity of real music repertoire, that these problems no longer hold substance.

What Is Ground Truth?

For tasks concerning linking music and metadata, we typically gather ground truth test data by having expert users annotate the music metadata. Machine listening and other automated methods are then challenged to match, as closely as possible, the annotations of the expert users. There appears to be an underlying assumption that the automated method will never equal or surpass the experts' capabilities despite that being the goal. However, results in our evaluation of real-time DTW methods (see Section 4.6) and our guitar tab based chord detection test (see Section 6.2.6) suggest that human annotators can be beaten. Work by Dixon et al. [2011] also raises similar questions about ground truth. We therefore argue the case that automatically produced data can be used to train other methods in cases where optimal solutions exist to train sub-optimal methods. Score aligned audio being used to test automatic annotation systems [Turetsky and Ellis, 2003] or onset detection [You and Dannenberg, 2007] are established examples. Other areas where automated data could be considered ground truth are in tagging applications where the ground truth has been automatically mined from the web, and singing/speech detection where the ground truth has been produced by automatically aligning lyrics with the music audio. Automated ground truth is necessary to allow us to use the large quantities of data that is available.

7.4 Future Research Questions

After examining the research questions laid out in Chapter 1 we have discovered many new avenues of research that can expand on our applications and methods. This work and others [Seltzer and Zhang, 2009], [Knees et al., 2005] & [Schedl, 2008] show that there are many opportunities to exploit the data deluge within MIR.

Further DTW Modifications

We expect the Greedy method to represent the minimal computation necessary to align two sequences. However, it may be possible to compute alignments quicker if the sequences were first built into a tree structure and alignments focussed only on the musical events akin to a suffix tree search.

WTW currently repeats the computation of overlapping windows within the cost matrix. Future improvements of WTW might focus on re-using areas of one window within the next. Additionally, having larger windows in WTW have been shown to increase the robustness of the path alignment to the detriment of performance. Therefore, a future improvement could use the path cost as a confidence estimator to dynamically adjust the size of the windows. The A-Star modification could lead to greater reductions in processing time if the cost estimation could be brought closer to the eventual window path cost. This might be possible with improved window guidance or by making an estimation and expanding this if and when the estimation stops the DTW reaching the edge of the window.

All of the modifications of DTW have low computational requirements and therefore could be extended using the multiple hypothesis approach by Arzt et al. [2008] where three or more paths are processed simultaneously starting from different alignment points. The selected path can be switched when another path attains a lower path cost.

Real-time DTW Enabled Applications

Our real-time modifications of DTW open up the possibility of unlimited sequence alignment. This could allow such applications as movie synchronisation, subtitle synchronisation, radio synchronisation, auto-cue synchronisation, as well as lead to constantly running sound and music identification.

GTT: Guitar Tab Restructuring

A possible extension of the Guitar Tab Toolkit would be to automatically restructure guitar tabs. This would allow a tab to be standardised for assisting with importing the tab into standard software like Guitar Tab Pro or for reducing the size of the tab to show a summarised

version on the screen of a smartphone. Additionally, restructuring would make it easier to compare tabs side by side when aligned using the Score Similarity function.

The Super Tab

Future work of the guitar tab analysis and ranking could involve comparing the separate elements of the tabs with other versions and then combining the most elements. For example, we could use the most common structural sequence, chord set, introduction, chorus, verse, lyrics, tablature, *etc* all separately and join them to make a super tab, greater than the sum of its parts.

Multi-modal Metadata Alignment

Within the various online music metadata, there are great opportunities in combining the different metadata modalities, such as lyrics, tabs, chord sequences, sheet music and videos to create a set of linked data. Furthermore, the various types of metadata can be used to re-construct the full song when parts of one are missing. For example, guitar tabs are rarely complete and usually contain snippets of lyrics. If these lyrics are aligned with a complete lyrics file then the tab can be restructured around this. This has been examined for the case of chord sequences and lyrics in [Mauch et al., 2010], however other metadata types can be combined to create new applications.

The Concurrence Factor

Concurrence has been shown to reliably indicate accuracy within chord sequences, structure sequences, and lyrics. Another ranking method might focus on combining concurrence with SERP Rank, User Rating, and metadata attributes to discover a set of weighted ranking methods. Future work should examine Concurrence in comparison with Multiple Classifier Systems. Future direction could be to examine the potential of Concurrence in ranking other types of metadata where there exist multiple variations for the same subject. Within music, we could rank the accuracy of guitar tablature notation and other transcriptions, chord definitions, or textual descriptors such as tags, artist spellings, *etc.* As the only requirements for the metadata are that people are annotating an event, there is potential beyond music for Concurrence to automatically rank the accuracy of ratings, reviews, and the correctness of academic references.

References

Paul E. Allen and Roger B. Dannenberg. Tracking Musical Beats in Real Time. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 140–143, 1990. 44

Willi Apel. *The Notation of Polyphonic Music, 900-1600*. Medieval Academy of America, 5th edition, 1961. 79

Vlora Arifi, Michael Clausen, Frank Kurth, and Meinard Müller. Automatic Synchronization of Music Data in Score-, MIDI- and PCM-Format. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, pages 219—220, 2003. 41

Vlora Arifi, Michael Clausen, Frank Kurth, and Meinard Müller. Score-PCM Music Synchronization Based on Extracted Score Parameters. In Uffe Kock Wiil, editor, *Proceedings of the 2nd International Symposium for Computer Music Modeling and Retrieval (CMMR)*, volume 3310, pages 193–210, 2004. 19, 41

Andreas Arzt and Gerhard Widmer. Towards Effective "Any-Time" Music Tracking. In *Proceedings of the Starting AI Researchers' Symposium (STAIRS)*, 2010. 38

Andreas Arzt, Gerhard Widmer, and Simon Dixon. Automatic Page Turning for Musicians via Real-Time Machine Listening. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, 2008. 41, 52, 134

Richard Baraniuk. More Is Less: Signal Processing and the Data Deluge. *Science*, 331(6018):717–719, February 2011. 17

Luke Barrington, Damien O'Malley, Douglas Turnbull, and Gert Lanckriet. User-Centered Design of a Social Game to Tag Music. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 7–10, New York, NY, USA, 2009. ACM. 29

Mathieu Barthet, Amélie Anglade, Gyorgy Fazekas, Sefki Kolozali, and Robert Macrae. Music Recommendation for Music Learning: Hotttabs, a Multimedia Guitar Tutor. In *Proceedings of the 2nd ACM RECSYS Workshop On Music Recommendation And Discovery (WOMRAD)*, 2011. 22, 101

S. Baumann and O. Hummel. Enhancing Music Recommendation Algorithms Using Cultural Metadata. *Journal of New Music Research*, 34(2):161–172, 2005. 26

Martin Beckmann. *Dynamic Programming of Economic Decisions*. Springer, Berlin, 1968. 30

Richard Bellman. *The Theory of Dynamic Programming*. Princeton University Press, 1957. 30

L. Bergroth, H. Hakonen, and T. Raita. A Survey of Longest Common Subsequence Algorithms. In *Proceedings of the 7th International Symposium on String Processing Information Retrieval (SPIRE)*, pages 39–48, 2000. 35

R. Berndt, I. Blümel, M. Clausen, D. Damm, J. Diet, D. Fellner, R. Klein, F. Krahl, M. Scherer, T. Schreck, I. Sens, R. Wessel, and Technische Universität Darmstadt. The PROBADO Project-Approach and Lessons Learned in Building a Digital Library System for Heterogeneous Non-textual Documents. In *Proceedings of the European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 376–383, 2010. 41

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, pages 591–596, 2011. 117, 132

Bir Bhanu and Xiaoli Zhou. Face Recognition from Face Profile Using Dynamic Time Warping. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)*, volume 4, pages 499–502, Washington, DC, USA, 2004. IEEE Computer Society. 19, 35

Al Borchers, Jon Herlocker, Joseph Konstan, and John Riedl. Ganging up on Information Overload. *Computer*, 31:106–108, April 1998. 26, 107

John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An Expert Ground-Truth Set for Audio Chord Recognition and Music Analysis. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, pages 633–638, 2011. 132

C. Myers and L. Rabiner and A. Rosenberg. An Investigation of the Use of Dynamic Time Warping for Word Spotting and Connected Speech Recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 5, pages 173–177, 1980. 19

Pedro Cano, Alex Loscos, and Jordi Bonada. Score-Performance Matching using HMMs. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 441–444, 1999. 19, 38

Chris Chafe, Bernard Mont-Reynaud, and Loren Rush. Towards an Intelligent Editor of Digital Audio: Recognition of Musical Constructs. *Computer Music Journal*, 6(1):30–41, 1982. 43

Wei Chai. *Automated Analysis of Musical Structure*. PhD thesis, Massachusetts Institute of Technology, 2005. 36

Heng Tze Cheng, Yi-Hsuan Yang, Yu-Ching Lin, and Homer H. Chen. Multimodal Structure Segmentation and Analysis of Music using Audio and Textual Information. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1677–1680, 2009. 35

T. Chesney. "Other People Benefit. I Benefit From Their Work." Sharing Guitar Tabs Online. *Journal of Computer Mediated Communication*, 10(1), November 2004. 80

David Clifford, Glenn Stone, Ivan Montoliu, Serge Rezzi, François-Pierre Martin, Philippe Guy, Stephen Bruce, and Sunil Kochhar. Alignment Using Variable Penalty Dynamic Time Warping. *Analytical Chemistry*, 81(3):1000–1007, January 2009. 19, 30, 35

Arshia Cont and Diemo Schwarz. Score Following at IRCAM. In *Music Information Retrieval Evaluation eXchange (MIREX)*, 2006. 19, 38, 74

Arshia Cont, Diemo Schwarz, Norbert Schnell, and Christopher Raphael. Evaluation of Real-Time Audio-to-Score Alignment. In *Proceedings of the 8th International Society for Music Information Retrieval (ISMIR)*, pages 315–316, 2007. 66, 68

N. Cook. Performance Analysis and Chopin's Mazurkas. *Musicae Scientae*, 11(2):183–205, 2007. 68

Florence Corpet. Multiple Sequence Alignment with Hierarchical Clustering. *Nucleic Acids Research*, 16(22):10881–10890, 1988. 39

Christoph Dalitz and Thomas Karsten. Using the Gamera Framework for Building a Lute Tablature Recognition System. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 478–481, 2005. 80

Christoph Dalitz and Christine Pranzas. German Lute Tablature Recognition. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR)*, pages 371–375, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3725-2. 80

David Damm, Christian Fremerey, Verena Thomas, and Michael Clausen. A Demonstration of the PROBADO Music System. In *Late Breaking Session of the 12th International Conference on Music Information Retrieval (ISMIR)*, 2011. URL `http://ismir2011.ismir.net/latebreaking/LB-6.pdf`. 41

R. B. Dannenberg. An On-Line Algorithm for Real-Time Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 193–198, 1984. 18, 30, 36, 41, 42, 90

Roger B. Dannenberg and Ning Hu. Polyphonic Audio Matching for Score Following and Intelligent Audio Editors. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 27–34, 2003. 36, 43

Roger B. Dannenberg, Marta Sanchez, Annabelle Joseph, Peter Capell, Robert Joseph, and Ronald Saul. A Computer-Based Multi-Media Tutor for Beginning Piano Students. *Journal of New Music Research*, 19(2-3):155–173, 1990. 42

Jürgen Diet and Frank Kurth. The PROBADO Music Repository at the Bavarian State Library. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 501–504, 2007. 41

Simon Dixon. Live Tracking of Musical Performances Using On-Line Time Warping. In *Proceedings of the 8th International Conference on Digital Audio Effects (DAFX)*, pages 92–97, Madrid, Spain, 2005. 12, 19, 35, 37, 48, 54, 58, 60

Simon Dixon and Gerhard Widmer. MATCH: A Music Alignment Tool Chest. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 492–497, 2005. 38, 58, 71

Simon Dixon, Dan Tidhar, and Emmanouil Benetos. The Temperament Police: The Truth, the Ground Truth, and Nothing but the Truth. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, 2011. 133

J. Stephen Downie, Kris West, Andreas Ehmann, and Emmanuel Vincent. The 2005 Music Information Retrieval Evaluation Exchange (MIREX 2005): Preliminary Overview. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 320–323, 2005. 68, 132

Alexander Duda, Andreas Nürnberger, and Sebastian Stober. Towards Query by Singing/Humming on Audio Databases. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 331–334, 2007. 32

Bryan Duggan, Brendan O'Shea, Mikel Gainza, and Padraig Cunningham. The Annotation of Traditional Irish Dance Music using MATT2 and TANSEY. In *Proceedings of the 8th Annual Information Technology & Telecommunication Conference (IT&T)*, 2009. 32

Sebastian Ewert, Meinard Müller, Daniel Müllensiefen, Michael Clausen, and Geraint A. Wiggins. Case Study "Beatles Songs" - What can be Learned from Unreliable Music Alignments? In Eleanor Selfridge-Field, Frans Wiering, and Geraint A. Wiggins, editors, *Knowledge Representation for Intelligent Music Processing*, Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum füer Informatik, Germany, 2009. 36

Chunsheng Fang. From Dynamic Time Warping (DTW) to Hidden Markov Model (HMM) Final project report for ECE742 Stochastic Decision Chunsheng Fang. *J Am Med Inform Assoc*, 4:1–7, 2009. 38

Jon Feldman, Ibrahim Abou-Faycal, and Matteo Frigo. A Fast Maximum-Likelihood Decoder for Convolutional Codes. In *Proceedings of the IEEE Semiannual Vehicular Technology Conference*, pages 371–375, 2002. 39

Ben Fields, Kurt Jacobson, Micheal Casey, and Mark Sandler. Do You Sound Like Your Friends? Exploring Artist Similarity via Artist Social Network Relationships and Audio Signal Processing. In *Proceedings of the International Computer Music Conference (ICMC)*, August 2008. 27, 28

Ben Fields, Kurt Jacobson, Christophe Rhodes, and Mark S. Analysis and Exploitation of Musician Social Networks for Recommendation and Discovery. *IEEE Transactions on Multimedia*, 13(4): 674–686, August 2011. 28

Scott Foster, W. Andrew Schloss, and A. Joseph Rockmore. Towards an Intelligent Editor of Digital Audio: Signal Processing Methods. *Computer Music Journal*, 6(1):42–51, 1982. 43

Hiromasa Fujihara, Masataka Goto, Jun Ogata, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G. Okuno. Automatic Synchronization between Lyrics and Music CD Recordings Based on Viterbi Alignment of Segregated Vocal Signals. In *Proceedings of the 8th IEEE International Symposium on Multimedia (ISM)*, pages 257–264, Washington, DC, USA, 2006. 27

M.J.F. Gales. Maximum Likelihood Linear Transformations for HMM-Based Speech Recognition. *Computer Speech and Language*, 12:75–98, 1998. 38

Francis Galton. Kinship and Correlation. *North American Review*, 150:419–431, 1890. 106

Silvia García-Díez, Marco Saerens, Mathieu Senelle, and François Fouss. A Simple-cycles Weighted Kernel Based on Harmony Structure for Similarity Retrieval. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, pages 61–66, 2011. 28

M. Goto and Y. Muraoka. A Beat Tracking System for Acoustic Signals of Music. In *Proceedings of the 2nd Annual ACM International Conference on Multimedia (ACM-MM)*, pages 365–372, New York, NY, United States, 1994. ACM. ISBN 0-89791-686-7. 44

M Goto and Y Muraoka. *A Real-time Beat Tracking System for Audio Signals*, pages 171–174. San Francisco: International Computer Music Association, 1995. 44

Maarten Grachten, Josep L. Arcos, and Ramon L. de Mantaras. Evolutionary Optimization of Music Performance Annotation. In *Proceedings of the 2nd International Symposium on Computer Music Modeling and Retrieval (CMMR)*, May 2004. 32

Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997. ISBN 0-521-58519-8. 34

C. Harte, M. Sandler, and S. Abdallah. Symbolic Representation Of Musical Chords: A Proposed Syntax For Text Annotations. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 66–71, 2005. 81

Christopher Harte, Mark Sandler, and Martin Gasser. Detecting Harmonic Change in Musical Audio. In *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia (AMCMM)*, pages 21–26, New York, NY, USA, 2006. ACM. 62

Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision Combination in Multiple Classifier Systems. *IEEE Transactions on Pattern Analysis ans Machine Intelligence*, 16(1), January 1994. 109

N. Hu, R. B. Dannenberg, and G. Tzanetakis. Polyphonic Audio Matching and Alignment for Music Retrieval. In *Proceedings of the Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 185–188, New Paltz, NY, October 2003. IEEE. 36

Nobumichi Ishimura, Takeshi Hashimoto, Shuichi Tsujimoto, and Suguru Arimoto. Spline Approximation of Line Images by Modified Dynamic Programming. *Systems and Computers in Japan*, 17: 21–29, 1986. 30

F. Itakura. Minimum Prediction Residual Principle Applied to Speech Recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):67–72, 1975. 19, 30, 37, 39, 48

Kurt Jacobson, Ben Fields, and Mark Sandler. Using Audio Analysis and Network Structure to Identify Communities in On-line Social Networks of Artists. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, 2008. 27, 28

Tristan Jehan, Michael Lew, and Cati Vaucelle. Cati Dance: Self-edited, Self-synchronized Music Video. In *Proceedings of the 30th International Conference on Computer Graphics and Interactice Techniques (SIGGRAPH)*, 2003. 18, 44

Min-Yen Kan, Ye Wang, Denny Iskandar, Tin Lay Nwe, and Arun Shenoy. LyricAlly: Automatic Synchronization of Textual Lyrics to Acoustic Music Signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):338–349, 2008. 18, 27, 43

Eamonn J. Keogh and Michael J. Pazzani. Scaling Up Dynamic Time Warping for Datamining Applications. In *Proceedings of the 6th International ACM Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 285–289, New York, NY, United States, 2000. 18, 19, 35

Holger Kirchhoff and Alexander Lerch. Evaluation of Features for Audio-to-Audio Alignment. *Journal of New Music Research*, 40(1):27–41, 2011. 36, 65, 66

Tetsuro Kitahara, Masataka Goto, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G Okuno. Instrogram: Probabilistic Representation of Instrument Existence for Polyphonic Music. *Information and Media Technologies*, 2(1):279–291, 2007. 38

J. Kittler and F. M. Alkoot. Sum versus vote fusion in multiple classifier systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25:110–115, January 2003. 109

Florian Kleedorfer, Peter Knees, and Tim Pohle. Oh Oh Oh Whoah! Towards Automatic Topic Detection In Song Lyrics. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 287–292, September 2008. 27

Peter Knees, Markus Schedl, and Gerhard Widmer. Multiple Lyrics Alignment: Automatic Retrieval of Song Lyrics. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 564–569, London, UK, September 2005. 27, 39, 117, 118, 133

Jan Korst and Gijs Geleijnse. Efficient Lyrics Retrieval and Alignment. In *Proceedings of the 3rd Philips Symposium on Intelligent Algorithms (SOIA)*, Eindhoven, the Netherlands, December 2006. 27, 117, 118

Frank Kurth, Meinard Müller, David Damm, Christian Fremerey, Andreas Ribbrock, and Michael Clausen. SYNCPLAYER — AN ADVANCED SYSTEM FOR MULTIMODAL MUSIC ACCESS. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, 2005. 41

Frank Kurth, Meinard Müller, Christian Fremerey, Yoon ha Chang, and Michael Clausen. Automated Synchronization of Scanned Sheet Music with Audio Recordings. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 261—266, 2007. 41

Edith L. M. Law, Luis von Ahn, Roger B. Dannenberg, and Mike Crawford. Tagatune: A Game for Music and Sound Annotation. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 535–538, 2007. 29

Benoít Legrand, C. S. Chang, S. H. Ong, Soek-Ying Neo, and Nallasivam Palanisamy. Chromosome Classification Using Dynamic Time Warping. *Pattern Recogn. Lett.*, 29(3):215–222, 2008. 35

Kjell Lemström and Sami Perttu. SEMEX - An Efficient Music Retrieval Prototype. In *Proceedings of the 1st International Symposium on Music Information Retrieval (ISMIR)*, pages 23–25, 2000. 32

Kjell Lemström and Esko Ukkonen. Including Interval Encoding into Edit Distance Based Music Comparison and Retrieval. In *Proceedings of the Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science (AISB)*, pages 53–60, 2000. 32

V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics*, 10(8):707–710, February 1966. 31

Rory A. Lewis, Wenxin Jiang, and Zbigniew W. Raś. Mining Scalar Representations in a Non-Tagged Music Database. In *Proceedings of the 17th International Conference on Foundations of Intelligent Systems (ISMIS)*, pages 445–454, Berlin, Heidelberg, 2008. Springer-Verlag. 32

Hwei-Jen Lin, Hung-Hsuan Wu, and Chun-Wei Wang. Music Matching Based on Rough Longest Common Subsequence. *Journal of information science and engineering*, 27(1):95–110, 2011. 35

Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 3540378812. 20, 27

Beth Logan, Andrew Kositsky, and Pedro Moreno. Semantic Analysis of Song Lyrics. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, volume 2, pages 827–830, Baltimore,Maryland, USA, 2004. 27

Julie Beth Lovins. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968. 118

R. Macrae and S. Dixon. A Guitar Tablature Score Follower. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 725–726, 2010a. 22, 101

Robert Macrae and Simon Dixon. Accurate Real-time Windowed Time Warping. In *Proceedings of the 11th International Conference on Music Information Retrieval (ISMIR)*, pages 423–428, 2010b. 22, 48, 54, 60

Robert Macrae and Simon Dixon. Guitar Tab Mining, Analysis and Ranking. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, pages 453–458, 2011. 21, 22, 28

Robert Macrae and Simon Dixon. From Toy to Tutor: Notescroller is a Game to Teach Music. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2008. 22

Robert Macrae, Xavier Anguera Miro, and Nuria Oliver. MuViSync: Realtime Music Video Alignment. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 534–539, 2010. 23, 66

Robert Macrae, Joachim Neumann, Xavier Anguera, Nuria Oliver, and Simon Dixon. Real-Time Synchronization of Multimedia Streams in a Mobile Device. In *Proceedings of the International Workshop on Advances in Music Information Research (AdMIRe)*, 2011. 23

Jose P. G. Mahedero, Álvaro MartÍnez, Pedro Cano, Markus Koppenberger, and Fabien Gouyon. Natural Language Processing of Lyrics. In *Proceedings of the 13th Annual ACM International Conference on Multimedia (ACM-MM)*, pages 475–478, New York, NY, USA, 2005. ACM. ISBN 1-59593-044-2. 28

Michael I Mandel and Daniel P W Ellis. A Web-based Game for Collecting Music Metadata. *Journal of New Music Research*, 37(2):151–165, 2008. 29

U. Marti and H. Bunke. Use of Positional Information in Sequence Alignment for Multiple Classifier Combination. In Josef Kittler and Fabio Roli, editors, *Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, pages 388–398. Springer Berlin / Heidelberg, 2001. ISBN 978-3-540-42284-6. 109

M. Mauch and S. Dixon. Simultaneous Estimation of Chords and Musical Context from Audio. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 18:1280–1289, 2010. 115

M. Mauch, C. Cannam, M. Davies, S. Dixon, C. Harte, S. Kolozali, D. Tidhar, and M. Sandler. OMRAS2 Metadata Project 2009. In *Proceedings of the 10th International Society for Music Information Retrieval (ISMIR)*, 2009. 81

Matthias Mauch. *Automatic Chord Transcription from Audio Using Computational Models of Musical Context*. PhD thesis, Queen Mary, University of London, 2010. 38

Matthias Mauch, Hiromasa Fujihara, and Masataka Goto. Lyrics-to-Audio Alignment and Phrase-Level Segmentation Using Incomplete Internet-Style Chord Annotations. In *Proceedings of the 7th Sound and Music Computing Conference (SMC)*, 2010. 27, 28, 38, 135

Rudolf Mayer and Andreas Rauber. Multimodal Aspects of Music Retrieval: Audio, Song Lyrics - and Beyond? In *Advances in Music Information Retrieval*, volume 274 of *Studies in Computational Intelligence*, pages 333–363. Springer Berlin / Heidelberg, 2010. 30

Rudolf Mayer, Robert Neumayer, and Andreas Rauber. Rhyme and Style Features for Musical Genre Classification by Song Lyrics. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, 2008. 27

Matt McVicar, Yizhao Ni, Raul Santos-Rodriguez, and Tijl De Bie. Leveraging Noisy Online Databases for use in Chord Recognition. In *Proceedings of the 12th International Society on Music Information Retrieval (ISMIR)*, pages 639–644, 2011a. 28, 115

Matt McVicar, Yizhao Ni, Raul Santos-Rodriguez, and Tijl De Bie. Using Online Chord Databases to Enhance Chord Recognition. *Journal of New Music Research*, 40(2):139–152, 2011b. 20, 28

Richard A. Medina. Content-based Indexing of Musical Scores. In *Proceedings of the 3rd ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 18–26, 2003. 34

Meinard Müller. *Information Retrieval for Music and Motion*. Springer-Verlag New York, Secaucus, NJ, USA, 2007. ISBN 3540740473. 19, 35

Meinard Müller and Daniel Appelt. Path-Constrained Partial Music Synchronization. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2008. 39

Meinard Müller and Frank Kurth. Enhancing Similarity Matrices for Music Audio Analysis. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 437–440, 2006. 41

Meinard Müller, Frank Kurth, and Tido Roder. Towards an Efficient Algorithm for Automatic Score-to-Audio Synchronization. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 365–372, 2004. 41

Meinard Müller, Frank Kurth, David Damm, Christian Fremerey, and Michael Clausen. Lyrics-Based Audio Retrieval and Multimodal Navigation in Music Collections. In *Proceedings of the 11th European Conference on Digital Libraries (ECDL)*, pages 112–123, 2007. 32

Meinard Müller, Masataka Goto, and Simon Dixon. Multimodal Music Processing (Dagstuhl Seminar 11041). *Dagstuhl Reports*, 1(1):68–101, 2011. 20

Hiroshi Murase and Rie Sakai. Moving Object Recognition in Eigenspace Representation: Gait Analysis and Lip Reading. *Pattern Recogn. Lett.*, 17(2):155–162, 1996. 35

C. Myers, L. Rabiner, and A. Rosenberg. Performance Tradeoffs in Dynamic Time Warping Algorithms for Isolated Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28 (6):623–635, 1980. 19, 20

S B Needleman and C D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. 39

Kimberly A. Neuendorf. *The Content Analysis Guidebook*. Sage Publications, 2002. 20

Tom O'Hara. Inferring the Meaning of Chord Sequences via Lyrics. In *Proceedings of the 2nd ACM RECSYS Workshop on Music Recommendation and Discovery (WOMRAD)*, 2011. 28

Nicola Orio and Diemo Schwarz. Alignment of Monophonic and Polyphonic Music to a Score. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 155–158, 2001. 36

Z. Ouyang and S. M. Shahidehpour. A Hybrid Artificial Neural Network-Dynamic Programming Approach to Unit Commitment. *IEEE Transactions on Power Systems*, 7(1):236–242, 1992. 30

F Pachet, G Westermann, and D Laigre. Musical Data Mining for Electronic Music Distribution. In *Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC)*, pages 101–106, Washington, DC, USA, 2001. IEEE Computer Society. 26

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, 1999. 27, 107, 109, 117

Bryan Pardo and William P. Birmingham. Query By Humming: How Good Can It Get. In *Proceedings of the Workshop on Music Information Retrieval (SIGIR)*, pages 71–111, 2003. 38

Karl Pearson. Contributions to the Mathematical Theory of Evolution, II: Skew Variation in Homogeneous Material. *Philosophical Transactions of the Royal Society of London*, 186:343–414, 1895. 106

Geoffroy Peeters. Sequence Representation of Music Structure using Higher-order Similarity Matrix and Maximum-likelihood Approach. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, 2007. 36

Laurent Pugin. Optical Music Recognition of Early Typographic Prints using Hidden Markov Models. In Kjell Lemström, Adam Tindale, and Roger Dannenberg, editors, *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR)*, pages 53–56, Victoria, Canada, October 2006. University of Victoria. 80

Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. ISBN 0-13-015157-2. 11, 49, 50

Lawrence R. Rabiner. *Readings in Speech Recognition*, chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-124-4. 38

Christopher Raphael. Music Plus One: A System for Flexible and Expressive Musical Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba, 2001. 18, 19, 38, 42

Chotirat Ann Ratanamahatana and Eamonn Keogh. Everything You Know About Dynamic Time Warping is Wrong. In *Proceedings of the ACM SIGKDD 3rd Workshop on Mining Temporal and Sequential Data*, 2004. 19

Seungmin Rho and Eenjun Hwang. FMF: Query Adaptive Melody Retrieval System. *Journal of Systems and Software*, 79:43–56, January 2006. 34

Shane Richmond. YouTube Users Uploading Two Days of Video Every Minute, 2011. URL `http://www.telegraph.co.uk/technology/google/8536634/YouTube-users-uploading-two-days-of-video-every-minute.html`. 17

Andrew Robertson and Mark Plumbley. B-Keeper: A Beat-Tracker for Live Performance. In *Proceedings of the 7th International Conference on New Interfaces for Musical Expression (NIME)*, pages 234–237, New York, NY, United States, 2007. 18, 44

Hiroaki Sakoe and Seibi Chiba. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):IEEE Transactions on Acoustics, Speech and Signal Processing In Acoustics, Speech and Signal Processing, IEEE Transactions on, Vol. 26, No. 1. (1978), pp. 43–49. Key: citeulike:3496861 IEEE Transactions on Acoustics, Speech and Signal Processing In Acoustics, Speech and Signal Processing, IEEE Transactions on, Vol. 26, No. 1. (1978), pp. 43–49. Key: citeulike:3496861 43–49, 1978. 19, 30, 35, 37, 39, 48, 58

Stan Salvador and Philip Chan. FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. In *Proceedings of the 3rd ACM SIGKDD Workshop on Mining Temporal and Sequential Data*, 2004. 15, 19, 37, 39, 48, 54, 74, 75

Craig Sapp. Comparative Analysis of Multiple Musical Performances. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 497–500, 2007. 68, 131

Markus Schedl. *Automatically Extracting, Analyzing, and Visualizing Information on Music Artists from the World Wide Web*. PhD thesis, Johannes Kepler University, Linz, Austria, July 2008. 20, 133

Markus Schedl, Peter Knees, Tim Pohle, and Gerhard Widmer. Towards an Automatically Generated Music Information System via Web Content Mining. In *Proceedings 30th Annual European Conference on Information Retrieval Research (ECIR)*, pages 585–590, Berlin, Heidelberg, 2008. Springer-Verlag. 27

Diemo Schwarz, Nicola Orio, and Norbert Schnell. Robust Polyphonic Midi Score Following with Hidden Markov Models. In *Proceedings of the 2004 International Computer Music Conference (ICMC)*, 2004. 19, 38

Michael L. Seltzer and Lei Zhang. The Data Deluge: Challenges and Opportunities of Unlimited Data in Statistical Signal Processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3701–3704, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-2353-8. 17, 133

J Serrà, E Gómez, P Herrera, and X Serra. Chroma Binary Similarity and Local Alignment Applied to Cover Song Identification. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(6): 1138–1151, 2008. 35

Kirk Skaugen. More Data Was Transmitted Over the Internet in 2010 Than All Previous Years Combined [VIDEO], 2011. URL `http://mashable.com/2011/10/20/kirk-skaugen-web-2/`. 17

Malcolm Slaney and William White. Similarity Based on Rating Data. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 479–484, Vienna, Austria, September 2007. Österreichische Computer Gesellschaft. 107

T F Smith and M S Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. 39

Stephen W. Smoliar, John A. Waterworth, and Peter R. Kellock. PianoFORTE: a system for Piano Education Beyond Notation Literacy. In *Proceedings of the third ACM international conference on Multimedia*, pages 457–465, New York, NY, USA, 1995. ACM. ISBN 0-89791-751-0. 42

Ferreol Soulez, Xavier Rodet, and Diemo Schwarz. Improving Polyphonic and Poly-Instrumental Music to Score Alignment. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, pages 143–148, 2003. 19, 36

Charles Spearman. General Intelligence. *American Journal of Psychology*, 15:201–293, 1904. 106

Michael Strube, Stefan Rapp, and Christoph Müller. The Influence of Minimum Edit Distance on Reference Resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 10, pages 312–319, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. 30

Xiaoyuan Su and Taghi M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, 2009(3):1–20, January 2009. 26

Iman S. H. Suyoto, Alexandra L. Uitdenbogerd, and Falk Scholer. Effective Retrieval of Polyphonic Audio with Polyphonic Symbolic Queries. In *Proceedings of the 9th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR)*, pages 105–114, New York, NY, USA, 2007. ISBN 978-1-59593-778-0. 34

Dacheng Tao, Hao Liu, and Xiaoou Tang. K-BOX: a Query-by-Singing Based Music Retrieval System. In *Proceedings of the 12th Annual ACM International Conference on Multimedia (ACM-MM)*, MULTIMEDIA '04, pages 464–467, New York, NY, USA, 2004. ACM. ISBN 1-58113-893-8. 36

Verena Thomas, Christian Fremerey, David Damm, and Michael Clausen. Slave: A Score-Lyrics-Audio-Video-Explorer. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, pages 717–722, 2009. 41

Karl Tillström. Java Exception Matching in Real Time using Fuzzy Logic. Master's thesis, University of Gothenburg, May 2010. 39

Derek Tingle, Youngmoo E. Kim, and Douglas Turnbull. Exploring automatic music annotation with "acoustically-objective" tags. In *Proceedings of the International ACM Conference on Multimedia Information Retrieval (MIR)*, pages 55–62, New York, NY, USA, 2010. 30

Robert J. Turetsky and Daniel P.W. Ellis. Ground-Truth Transcriptions of Real Music from Force-Aligned MIDI Syntheses. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, pages 135–141, 2003. 18, 19, 20, 35, 43, 66, 69, 133

Julián Urbano. Information Retrieval Meta-evaluation: Challenges and Opportunities in the Music Domain. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, pages 609–614, 2011. 132

Vladimir Viro. PEACHNOTE: Music Score Search and Analysis Platform. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, pages 359–362, 2011. 18, 43

Andrew Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. 38

Luis von Ahn and Laura Dabbish. Labeling Images with a Computer Game. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI)*, pages 319–326, New York, NY, USA, 2004. ACM. 29

Luis von Ahn, Manuel Blum, Nicholas Hopper, and John Langford. CAPTCHA: Using Hard AI Problems for Security. In Eli Biham, editor, *Advances in Cryptology: EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 646–646. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-14039-9. 29

H. J. L. M. Vullings, M. H. G. Verhaegen, and H. B. Verbruggen. Automated ECG Segmentation With Dynamic Time Warping. In *Proceedings of the 20th Annual International Conference of the IEEE In Engineering in Medicine and Biology Society*, volume 1, pages 163–166, 1998. 19, 30, 35

Avery Wang. The Shazam Music Recognition Service. *Communications of the ACM*, 49:44–48, August 2006. 43

Ye Wang, Min-Yen Kan, Tin Lay Nwe, Arun Shenoy, and Jun Yin. LyricAlly: Automatic Synchronization of Acoustic Musical Signals and Textual Lyrics. In *Proceedings of the 12th Annual ACM International Conference on Multimedia (ACM-MM)*, pages 212–219, New York, NY, USA, 2004. ISBN 1-58113-893-8. 18, 27, 38

Lee Ling Wei, Q.A. Salih, and Ho Sooi Hock. Optical Tablature Recognition (OTR) System: Using Fourier Descriptors as a Recognition Tool. In *Proceedings of the International Conference on Audio, Language and Image Processing (ICALIP)*, pages 1532–1539, 2008. 80

Chi Wong, Wai Szeto, and Kin Wong. Automatic Lyrics Alignment for Cantonese Popular Music. *Multimedia Systems*, 12:307–323, 2007. 27, 36

Xiao Wu, Ming Li, Jian Liu, Jun Yang, and Yonghong Yan. A Top-down Approach to Melody Match in Pitch Contour for Query by Humming. In *Proceedings of the 5th International Symposium Chinese Spoken Language Processing*, pages 669–680, 2006. 34

Linxing Xiao, Jie Zhou, and Tong Zhang. Using DTW Based Unsupervised Segmentation to Improve the Vocal Part Detection in Pop Music. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 1193–1196, 2008. 36

Fan Yang. Adaptive Music Recommendation System. Master's thesis, Massachusetts Institute of Technology, June 2010. 30

Wei You and Roger B. Dannenberg. Polyphonic Music Note Onset Detection using Semi-Supervised Learning. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, 2007. 20, 43, 133

Mark Zadel and Ichiro Fujinaga. Web Services for Music Information Retrieval. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 478–483. Universitat Pompeu Fabra, 2004. 27

Sven Meyer zu Eissen and Benno Stein. Genre Classification of Web Pages: User Study and Feasibility Analysis. In *Advances in Artificial Intelligence*, pages 256–269. Springer, 2004. 27, 32

A

Guitar Tablature Chord List

This list was taken from ultimate_guitar.com. There are some errors (such as A5, Ab/F, Bsus, D...) but as this is the most commonly used online chord dictionary, it was used as it is in our tests.

[A, AM, Amaj] [A, C#, E] [0 0 2 2 2 0]
[A#5, Bb5] [A, C#, F] [x 0 3 2 2 1]
[A/Ab] [A, C#, E, G#] [x 0 2 1 2 0]
[A/B] [A, B, C#, E] [0 0 2 4 2 0]
[A/D] [A, C#, D, E] [x 0 0 2 2 0]
[A/G] [A, C#, E, G] [3 x 2 2 2 0]
[A/Gb] [A, C#, E, F#] [0 0 2 2 2 2]
[A5] [A, E] [5 7 7 x x 5]
[A6] [A, C#, E, F#] [0 0 2 2 2 2]
[A6/7] [A, C#, E, F#, G] [0 0 2 0 2 2]
[A6/7sus] [A, D, E, F#, G] [5 5 4 0 3 0]
[A7] [A, C#, E, G] [3 x 2 2 2 0]
[A7(#5)] [A, C#, F, G] [1 0 3 0 2 1]
[A7/add11] [A, C#, D, E, G] [x 0 0 0 2 0]
[A7sus4] [A, D, E, G] [x 0 2 0 3 0]
[Aadd9] [A, B, C#, E] [0 0 2 4 2 0]
[Aaug/D] [A, C#, D, F] [x x 0 2 2 1]
[Aaug/G] [A, C#, F, G] [1 0 3 0 2 1]
[Ab, G#, AbM, Abmaj, G#M, G#maj] [C, D#, G#] [4 6 6 5 4 4]
[Ab#5] [C, E, G#] [x 3 2 1 1 0]
[Ab/A, G#/A] [A, C, D#, G#] [x x 1 2 1 4]
[Ab/F, G#/F] [C, D#, F, G#] [x 8 10 8 9 8]
[Ab/Gb, G#/Gb] [C, D#, F#, G#] [x x 1 1 1 2]
[Ab5, G#5] [D#, G#] [4 6 6 x x 4]
[Ab6, G#6] [C, D#, F, G#] [x 8 10 8 9 8]
[Ab7, G#7] [C, D#, F#, G#] [x x 1 1 1 2]
[Abdim/E, G#dim/E] [B, D, E, G#] [0 2 0 1 0 0]
[Abdim/Eb, G#dim/Eb] [B, D, D#, G#] [x x 0 4 4 4]
[Abdim/F, G#dim/F] [B, D, F, G#] [x 2 0 1 0 1]
[Abdim7, G#dim7] [B, D, F, G#] [x 2 0 1 0 1]
[Abm, G#m, Abmin, G#min] [B, D#, G#] [x x 6 4 4 4]
[Abm/D, G#m/D] [B, D, D#, G#] [x x 0 4 4 4]
[Abm/E, G#m/E] [B, D#, E, G#] [0 2 1 1 0 0]
[Abm/Gb, G#m/Gb] [B, D#, F#, G#] [x x 4 4 4 4]
[Abm7, G#m7] [B, D#, F#, G#] [x x 4 4 4 4]
[Absus, G#sus] [C#, D#, G#] [x x 6 6 4 4]
[Absus2/F, G#sus2/F] [A#, D#, F, G#] [x 1 3 1 4 1]
[Adim/Ab] [A, C, D#, G#] [x x 1 2 1 4]
[Adim/E] [A, C, D#, E] [0 3 x 2 4 0]
[Adim/F] [A, C, D#, F] [x x 1 2 1 1]
[Adim/G] [A, C, D#, G] [x x 1 2 1 3]
[Adim/Gb] [A, C, D#, F#] [x x 1 2 1 2]
[Adim7] [A, C, D#, F#] [x x 1 2 1 2]
[Am, Amin] [A, C, E] [x 0 2 2 1 0]
[Am/B] [A, B, C, E] [0 0 7 5 0 0]
[Am/D] [A, C, D, E] [x x 0 2 1 0]
[Am/Eb] [A, C, D#, E] [0 3 x 2 4 0]
[Am/F] [A, C, E, F] [0 0 3 2 1 0]
[Am/G] [A, C, E, G] [0 0 2 0 1 3]
[Am/Gb] [A, C, E, F#] [x 0 2 2 1 2]
[Am6] [A, C, E, F#] [x 0 2 2 1 2]
[Am7] [A, C, E, G] [0 0 2 0 1 3]
[Am7(b5)] [A, C, D#, G] [x x 1 2 1 3]
[Am7/add11] [A, C, D, E, G] [x 5 7 5 8 0]
[Amaj7, AM7] [A, C#, E, G#] [x 0 2 1 2 0]
[Amin/maj9, Amin/M9, Am/maj9] [A, B, C, E, G#] [x 0 6 5 5 7]
[Asus] [A, D, E] [0 0 2 2 3 0]
[Asus2] [A, B, E] [0 0 2 2 0 0]
[Asus2/Ab] [A, B, E, G#] [x 0 2 1 0 0]
[Asus2/C] [A, B, C, E] [0 0 7 5 0 0]
[Asus2/D] [A, B, D, E] [0 2 0 2 0 0]
[Asus2/Db] [A, B, C#, E] [0 0 2 4 2 0]
[Asus2/Eb] [A, B, D#, E] [x 2 1 2 0 0]
[Asus2/F] [A, B, E, F] [0 0 3 2 0 0]
[Asus2/G] [A, B, E, G] [x 0 2 0 0 0]
[Asus2/Gb] [A, B, E, F#] [x 0 4 4 0 0]
[Asus4/Ab] [A, D, E, G#] [4 x 0 2 3 0]
[Asus4/B] [A, B, D, E] [0 2 0 2 0 0]
[Asus4/Bb] [A, A#, D, E] [0 1 x 2 3 0]
[Asus4/C] [A, C, D, E] [x x 0 2 1 0]
[Asus4/Db] [A, C#, D, E] [x 0 0 2 2 0]
[Asus4/F] [A, D, E, F] [x x 7 7 6 0]
[Asus4/G] [A, D, E, G] [x 0 2 0 3 0]
[Asus4/Gb] [A, D, E, F#] [0 0 0 2 3 2]
[B, BM, Bmaj] [B, D#, F#] [x 2 4 4 4 2]
[B#5, Cb5] [B, D#, G] [3 2 1 0 0 3]
[B/A] [A, B, D#, F#] [2 x 1 2 0 2]
[B/Ab] [B, D#, F#, G#] [x x 4 4 4 4]
[B/E] [B, D#, E, F#] [x 2 2 4 4 2]
[B5] [B, F#] [x 2 4 4 x 2]
[B6] [B, D#, F#, G#] [x x 4 4 4 4]
[B7] [A, B, D#, F#] [2 x 1 2 0 2]
[B7/add11] [A, B, D#, E, F#] [0 0 4 4 4 0]
[B7sus4] [A, B, E, F#] [x 0 4 4 0 0]
[Baug/E] [B, D#, E, G] [3 x 1 0 0 0]
[Bb, A#, BbM, Bbmaj, A#M, A#maj] [A#, D, F] [1 1 3 3 3 1]
[Bb#5] [A#, D, F#] [x x 0 3 3 2]
[Bbb5, A#b5] [A#, D, E] [x x 0 3 x 0]
[Bb/A, A#/A] [A, A#, D, F] [1 1 3 2 3 1]
[Bb/Ab, A#/Ab] [A#, D, F, G#] [x 1 3 1 3 1]
[Bb/Db, A#/Db] [A#, C#, D, F] [x x 0 6 6 6]

[Bb/E, A#/E] [A#, D, E, F] [x 1 3 3 3 0]
[Bb/G, A#/G] [A#, D, F, G] [3 5 3 3 3 3]
[Bb5, A#5] [A#, F] [6 8 8 x x 6]
[Bb6, A#6] [A#, D, F, G] [3 5 3 3 3 3]
[Bb6/add9, A#6/add9] [A#, C, D, F, G] [x 3 3 3 3 3]
[Bb7, A#7] [A#, D, F, G#] [x 1 3 1 3 1]
[Bb7sus4, A#7sus4] [A#, D#, F, G#] [x 1 3 1 4 1]
[Bbadd#11, A#add#11] [A#, D, E, F] [x 1 3 3 3 0]
[Bbaug/E, A#aug/E] [A#, D, E, F#] [2 x 4 3 3 0]
[Bbdim/C, A#dim/C] [A#, C, C#, E] [x 3 x 3 2 0]
[Bbdim/D, A#dim/D] [A#, C#, D, E] [x x 0 3 2 0]
[Bbdim/G, A#dim/G] [A#, C#, E, G] [x 1 2 0 2 0]
[Bbdim/Gb, A#dim/Gb] [A#, C#, E, F#] [2 4 2 3 2 2]
[Bbdim7, A#dim7] [A#, C#, E, G] [x 1 2 0 2 0]
[Bbm, A#m, Bbmin, A#min] [A#, C#, F] [1 1 3 3 2 1]
[Bbm/Ab, A#m/Ab] [A#, C#, F, G#] [x 1 3 1 2 1]
[Bbm/D, A#m/D] [A#, C#, D, F] [x x 0 6 6 6]
[Bbm/Gb, A#m/Gb] [A#, C#, F, F#] [x x 3 3 2 2]
[Bbm7, A#m7] [A#, C#, F, G#] [x 1 3 1 2 1]
[Bbmaj7, A#maj7, BbM7, A#M7] [A, A#, D, F] [1 1 3 2 3 1]
[Bbmaj9, A#maj9, BbM9, A#M9] [A, A#, C, D, F] [x 3 3 3 3 5]
[Bbsus2, A#sus2] [A#, C, F] [x x 3 3 1 1]
[Bbsus2/G, A#sus2/G] [A#, C, F, G] [x 3 5 3 6 3]
[Bbsus4/Ab, A#sus4/Ab] [A#, D#, F, G#] [x 1 3 1 4 1]
[Bdim/A] [A, B, D, F] [1 2 3 2 3 1]
[Bdim/Ab] [B, D, F, G#] [x 2 0 1 0 1]
[Bdim/G] [B, D, F, G] [1 x 0 0 0 3]
[Bdim7] [B, D, F, G#] [x 2 0 1 0 1]
[Bm, Bmin] [B, D, F#] [2 2 4 4 3 2]
[Bm/A] [A, B, D, F#] [x 0 4 4 3 2]
[Bm/G] [B, D, F#, G] [2 2 0 0 0 3]
[Bm7] [A, B, D, F#] [x 0 4 4 3 2]
[Bm7(b5)] [A, B, D, F] [1 2 3 2 3 1]
[Bm7/add11] [A, B, D, E, F#] [0 0 2 4 3 2]
[Bmaj7/#11, BM7/#11] [A#, B, D#, F, F#] [x 2 3 3 4 2]
[Bsus] [B, E, F#] [7 9 9 x x 0]
[Bsus2] [B, C#, F#] [x 4 4 4 x 2]
[Bsus2/E] [B, C#, E, F#] [x 4 4 4 x 0]
[Bsus4/A] [A, B, E, F#] [x 0 4 4 0 0]
[Bsus4/Ab] [B, E, F#, G#] [0 2 2 1 0 2]
[Bsus4/Db] [B, C#, E, F#] [x 4 4 4 x 0]
[Bsus4/Eb] [B, D#, E, F#] [x 2 2 4 4 2]
[Bsus4/G] [B, E, F#, G] [0 2 2 0 0 2]
[C, CM, Cmaj] [C, E, G] [0 3 2 0 1 0]
[C#5, Db5] [C, E, G#] [x 3 2 1 1 0]
[C/A] [A, C, E, G] [0 0 2 0 1 3]
[C/B] [B, C, E, G] [0 3 2 0 0 0]
[C/Bb] [A#, C, E, G] [x 3 5 3 5 3]
[C/D] [C, D, E, G] [3 x 0 0 1 0]
[C/F] [C, E, F, G] [x 3 3 0 1 0]
[C5] [C, G] [x 3 5 5 x 3]
[C6] [A, C, E, G] [0 0 2 0 1 3]
[C6/add9] [A, C, D, E, G] [x 5 7 5 8 0]
[C7] [A#, C, E, G] [x 3 5 3 5 3]
[C7sus4] [A#, C, F, G] [x 3 5 3 6 3]
[C9(b5)] [A#, C, D, E, F#] [0 3 x 3 3 2]
[Cadd9] [C, D, E, G] [3 x 0 0 1 0]
[Cdim/A] [A, C, D#, F#] [x x 1 2 1 2]
[Cdim/Ab] [C, D#, F#, G#] [x x 1 1 1 2]
[Cdim/D] [C, D, D#, F#] [x 5 4 5 4 2]
[Cdim7] [A, C, D#, F#] [x x 1 2 1 2]
[Cm, Cmin] [C, D#, G] [x 3 5 5 4 3]
[Cm/A] [A, C, D#, G] [x x 1 2 1 3]
[Cm/Bb] [A#, C, D#, G] [x 3 5 3 4 3]
[Cm6] [A, C, D#, G] [x x 1 2 1 3]
[Cm7] [A#, C, D#, G] [x 3 5 3 4 3]
[Cmaj7, CM7] [B, C, E, G] [0 3 2 0 0 0]
[Cmaj9, CM9] [B, C, D, E, G] [x 3 0 0 0 0]
[Csus] [C, F, G] [x 3 3 0 1 1]
[Csus2] [C, D, G] [x 5 5 5 x 3]
[Csus2/A] [A, C, D, G] [x x 0 2 1 3]
[Csus2/B] [B, C, D, G] [3 3 0 0 0 3]
[Csus2/E] [C, D, E, G] [3 x 0 0 1 0]
[Csus2/F] [C, D, F, G] [3 3 0 0 1 1]
[Csus4/A] [A, C, F, G] [3 x 3 2 1 1]
[Csus4/B] [B, C, F, G] [x 3 3 0 0 3]
[Csus4/Bb] [A#, C, F, G] [x 3 5 3 6 3]
[Csus4/D] [C, D, F, G] [3 3 0 0 1 1]
[Csus4/E] [C, E, F, G] [x 3 3 0 1 0]
[D, DM, Dmaj] [A, D, F#] [x 5 4 2 3 2]
[D#5, Eb5] [A#, D, F#] [x x 0 3 3 2]
[D/B] [A, B, D, F#] [x 0 4 4 3 2]
[D/C] [A, C, D, F#] [x 0 0 2 1 2]
[D/Db] [A, C#, D, F#] [x x 0 2 2 2]
[D/E] [A, D, E, F#] [0 0 0 2 3 2]
[D/G] [A, D, F#, G] [5 x 4 0 3 5]
[D5] [A, D] [5 5 7 7 x 5]
[D6] [A, B, D, F#] [x 0 4 4 3 2]
[D6/add9] [A, B, D, E, F#] [0 0 2 4 3 2]
[D7] [A, C, D, F#] [x 0 0 2 1 2]
[D7sus4] [A, C, D, G] [x x 0 2 1 3]
[D9] [A, C, D, E, F#] [0 0 0 2 1 2]
[D9(#5)] [A#, C, D, E, F#] [0 3 x 3 3 2]
[Dadd9] [A, D, E, F#] [0 0 0 2 3 2]
[Daug/E] [A#, D, E, F#] [2 x 4 3 3 0]
[Db, C#, DbM, Dbmaj, C#M, C#maj] [C#, F, G#] [4 4 6 6 6 4]
[Db#5] [A, C#, F] [x 0 3 2 2 1]
[Dbb5, C#b5] [C#, F, G] [x x 3 0 2 1]
[Db/B, C#/B] [B, C#, F, G#] [x 4 3 4 0 4]
[Db/Bb, C#/Bb] [A#, C#, F, G#] [x 1 3 1 2 1]
[Db/C, C#/C] [C, C#, F, G#] [x 3 3 1 2 1]
[Db5, C#5] [C#, G#] [x 4 6 6 x 4]
[Db6, C#6] [A#, C#, F, G#] [x 1 3 1 2 1]
[Db7, C#7] [B, C#, F, G#] [x 4 3 4 0 4]
[Dbaug/D, C#aug/D] [A, C#, D, F] [x x 0 2 2 1]
[Dbaug/G, C#aug/G] [A, C#, F, G] [1 0 3 0 2 1]
[Dbdim/A, C#dim/A] [A, C#, E, G] [3 x 2 2 2 0]
[Dbdim/B, C#dim/B] [B, C#, E, G] [0 2 2 0 2 0]
[Dbdim/Bb, C#dim/Bb] [A#, C#, E, G] [x 1 2 0 2 0]
[Dbdim/D, C#dim/D] [C#, D, E, G] [3 x 0 0 2 0]
[Dbdim7, C#dim7] [A#, C#, E, G] [x 1 2 0 2 0]
[Dbm, C#m, Dbmin, C#min] [C#, E, G#] [x 4 6 6 5 4]
[Dbm/A, C#m/A] [A, C#, E, G#] [x 0 2 1 2 0]
[Dbm/B, C#m/B] [B, C#, E, G#] [0 2 2 1 2 0]

[Dbm7, C#m7] [B, C#, E, G#] [0 2 2 1 2 0]
[Dbm7(b5), C#m7(b5)] [B, C#, E, G] [0 2 2 0 2 0]
[Dbmaj7, C#maj7, DbM7, C#M7] [C, C#, F, G#] [x 3 3 1 2 1]
[Dbsus2, C#sus2] [C#, D#, G#] [x x 6 6 4 4]
[Dbsus4/Bb, C#sus4/Bb] [A#, C#, F#, G#] [x x 4 3 2 4]
[Ddim/B] [B, D, F, G#] [x 2 0 1 0 1]
[Ddim/Bb] [A#, D, F, G#] [x 1 3 1 3 1]
[Ddim/C] [C, D, F, G#] [x x 0 1 1 1]
[Ddim7] [B, D, F, G#] [x 2 0 1 0 1]
[Dm, Dmin] [A, D, F] [x 0 0 2 3 1]
[Dm/B] [A, B, D, F] [1 2 3 2 3 1]
[Dm/Bb] [A, A#, D, F] [1 1 3 2 3 1]
[Dm/C] [A, C, D, F] [x 5 7 5 6 5]
[Dm/Db] [A, C#, D, F] [x x 0 2 2 1]
[Dm/E] [A, D, E, F] [x x 7 7 6 0]
[Dm6] [A, B, D, F] [x 2 0 2 0 1]
[Dm7] [A, C, D, F] [x 5 7 5 6 5]
[Dm7(b5)] [C, D, F, G#] [x x 0 1 1 1]
[Dm7/add11] [A, C, D, F, G] [3 x 0 2 1 1]
[Dmaj7, DM7] [A, C#, D, F#] [x x 0 14 14 14]
[Dmin/maj7, Dmin/M7, Dm/maj7] [A, C#, D, F] [x x 0 2 2 1]
[Dsus] [A, D, G] [5 x 0 0 3 5]
[Dsus2] [A, D, E] [5 5 7 7 x 0]
[Dsus2/Ab] [A, D, E, G#] [4 x 0 2 3 0]
[Dsus2/B] [A, B, D, E] [0 2 0 2 0 0]
[Dsus2/Bb] [A, A#, D, E] [0 1 x 2 3 0]
[Dsus2/C] [A, C, D, E] [x x 0 2 1 0]
[Dsus2/Db] [A, C#, D, E] [x 0 0 2 2 0]
[Dsus2/F] [A, D, E, F] [x x 7 7 6 0]
[Dsus2/G] [A, D, E, G] [x 0 2 0 3 0]
[Dsus2/Gb] [A, D, E, F#] [0 0 0 2 3 2]
[Dsus4/B] [A, B, D, G] [3 0 0 0 0 3]
[Dsus4/C] [A, C, D, G] [x x 0 2 1 3]
[Dsus4/E] [A, D, E, G] [x 0 2 0 3 0]
[Dsus4/Gb] [A, D, F#, G] [5 x 4 0 3 5]
[E, EM, Emaj] [B, E, G#] [0 2 2 1 0 0]
[E#5, Fb5] [C, E, G#] [x 3 2 1 1 0]
[E/A] [A, B, E, G#] [x 0 2 1 0 0]
[E/D] [B, D, E, G#] [0 2 0 1 0 0]
[E/Db] [B, C#, E, G#] [0 2 2 1 2 0]
[E/Eb] [B, D#, E, G#] [0 2 1 1 0 0]
[E/Gb] [B, E, F#, G#] [0 2 2 1 0 2]
[E11/b9] [A, B, D, E, F, G#] [0 0 3 4 3 4]
[E5] [B, E] [0 2 x x x 0]
[E6] [B, C#, E, G#] [0 2 2 1 2 0]
[E7] [B, D, E, G#] [0 2 0 1 0 0]
[E7/add11] [A, B, D, E, G#] [x 0 0 1 0 0]
[E7/b9(b5)] [A#, D, E, F, G#] [0 1 3 1 3 1]
[E7sus4] [A, B, D, E] [0 2 0 2 0 0]
[E9] [B, D, E, F#, G#] [0 2 0 1 0 2]
[Eadd9] [B, E, F#, G#] [0 2 2 1 0 2]
[Eb, D#, EbM, Ebmaj, D#M, D#maj] [A#, D#, G] [x 1 1 3 4 3]
[Eb#5] [B, D#, G] [3 2 1 0 0 3]
[Eb/C, D#/C] [A#, C, D#, G] [x 3 5 3 4 3]
[Eb/D, D#/D] [A#, D, D#, G] [x 6 8 7 8 6]
[Eb/Db, D#/Db] [A#, C#, D#, G] [x 1 1 3 2 3]
[Eb/E, D#/E] [A#, D#, E, G] [x x 5 3 4 0]
[Eb5, D#5] [A#, D#] [x 6 8 8 x 6]
[Eb6, D#6] [A#, C, D#, G] [x 3 5 3 4 3]
[Eb7, D#7] [A#, C#, D#, G] [x 1 1 3 2 3]
[Ebaug/E, D#aug/E] [B, D#, E, G] [3 x 1 0 0 0]
[Ebdim/B, D#dim/B] [A, B, D#, F#] [2 x 1 2 0 2]
[Ebdim/C, D#dim/C] [A, C, D#, F#] [x x 1 2 1 2]
[Ebdim7, D#dim7] [A, C, D#, F#] [x x 1 2 1 2]
[Ebm, D#m, Ebmin, D#min] [A#, D#, F#] [x x 4 3 4 2]
[Ebm/Db, D#m/Db] [A#, C#, D#, F#] [x x 1 3 2 2]
[Ebm7, D#m7] [A#, C#, D#, F#] [x x 1 3 2 2]
[Ebmaj7, D#maj7, EbM7, D#M7] [A#, D, D#, G] [x 6 8 7 8 6]
[Ebsus2/Ab, D#sus2/Ab] [A#, D#, F, G#] [x 1 3 1 4 1]
[Ebsus4/F, D#sus4/F] [A#, D#, F, G#] [x 1 3 1 4 1]
[Edim/C] [A#, C, E, G] [x 3 5 3 5 3]
[Edim/D] [A#, D, E, G] [3 x 0 3 3 0]
[Edim/Db] [A#, C#, E, G] [x 1 2 0 2 0]
[Edim/Eb] [A#, D#, E, G] [x x 5 3 4 0]
[Edim7] [A#, C#, E, G] [x 1 2 0 2 0]
[Em, Emin] [B, E, G] [0 2 2 0 0 0]
[Em/A] [A, B, E, G] [3 x 2 2 0 0]
[Em/C] [B, C, E, G] [0 3 2 0 0 0]
[Em/D] [B, D, E, G] [0 2 0 0 0 0]
[Em/Db] [B, C#, E, G] [0 2 2 0 2 0]
[Em/Eb] [B, D#, E, G] [3 x 1 0 0 0]
[Em/Gb] [B, E, F#, G] [0 2 2 0 0 2]
[Em6] [B, C#, E, G] [0 2 2 0 2 0]
[Em7] [B, D, E, G] [0 2 0 0 0 0]
[Em7(b5)] [A#, D, E, G] [3 x 0 3 3 0]
[Em7/add11] [A, B, D, E, G] [0 0 0 0 0 0]
[Em9] [B, D, E, F#, G] [0 2 0 0 0 2]
[Emaj7, EM7] [B, D#, E, G#] [0 2 1 1 0 0]
[Emaj9, EM9] [B, D#, E, F#, G#] [0 2 1 1 0 2]
[Emin/maj7, Emin/M7, Em/maj7] [B, D#, E, G] [3 x 1 0 0 0]
[Emin/maj9, Emin/M9, Em/maj9] [B, D#, E, F#, G] [0 6 4 0 0 0]
[Esus] [A, B, E] [0 0 2 2 0 0]
[Esus2] [B, E, F#] [7 9 9 x x 0]
[Esus2/A] [A, B, E, F#] [x 0 4 4 0 0]
[Esus2/Ab] [B, E, F#, G#] [0 2 2 1 0 2]
[Esus2/Db] [B, C#, E, F#] [x 4 4 4 x 0]
[Esus2/Eb] [B, D#, E, F#] [x 2 2 4 4 2]
[Esus2/G] [B, E, F#, G] [0 2 2 0 0 2]
[Esus4/Ab] [A, B, E, G#] [x 0 2 1 0 0]
[Esus4/C] [A, B, C, E] [0 0 7 5 0 0]
[Esus4/D] [A, B, D, E] [0 2 0 2 0 0]
[Esus4/Db] [A, B, C#, E] [0 0 2 4 2 0]
[Esus4/Eb] [A, B, D#, E] [x 2 1 2 0 0]
[Esus4/F] [A, B, E, F] [0 0 3 2 0 0]
[Esus4/G] [A, B, E, G] [3 x 2 2 0 0]
[Esus4/Gb] [A, B, E, F#] [x 0 4 4 0 0]
[F, FM, Fmaj] [A, C, F] [1 3 3 2 1 1]
[F#5, Gb5] [A, C#, F] [x 0 3 2 2 1]
[F/D] [A, C, D, F] [x 5 7 5 6 5]
[F/E] [A, C, E, F] [0 0 3 2 1 0]
[F/Eb] [A, C, D#, F] [x x 1 2 1 1]
[F/G] [A, C, F, G] [3 x 3 2 1 1]
[F5] [C, F] [1 3 3 x x 1]
[F6] [A, C, D, F] [x 5 7 5 6 5]

[F6/add9] [A, C, D, F, G] [3 x 0 2 1 1]
[F7] [A, C, D#, F] [x x 1 2 1 1]
[Fadd9] [A, C, F, G] [3 x 3 2 1 1]
[Faug/D] [A, C#, D, F] [x x 0 2 2 1]
[Faug/G] [A, C#, F, G] [1 0 3 0 2 1]
[Fdim/D] [B, D, F, G#] [x 2 0 1 0 1]
[Fdim/Db] [B, C#, F, G#] [x 4 3 4 0 4]
[Fdim7] [B, D, F, G#] [x 2 0 1 0 1]
[Fm, Fmin] [C, F, G#] [x 3 3 1 1 1]
[Fm/D] [C, D, F, G#] [x x 0 1 1 1]
[Fm/Db] [C, C#, F, G#] [x 3 3 1 2 1]
[Fm/Eb] [C, D#, F, G#] [x 8 10 8 9 8]
[Fm6] [C, D, F, G#] [x x 0 1 1 1]
[Fm7] [C, D#, F, G#] [x 8 10 8 9 8]
[Fmaj7, FM7] [A, C, E, F] [0 0 3 2 1 0]
[Fmaj7/#11, FM7/#11] [A, B, C, E, F] [0 2 3 2 1 0]
[Fmaj9, FM9] [A, C, E, F, G] [0 0 3 0 1 3]
[Fsus] [A#, C, F] [x x 3 3 1 1]
[Fsus2] [C, F, G] [x 3 3 0 1 1]
[Fsus2/A] [A, C, F, G] [3 x 3 2 1 1]
[Fsus2/B] [B, C, F, G] [x 3 3 0 0 3]
[Fsus2/Bb] [A#, C, F, G] [x 3 5 3 6 3]
[Fsus2/D] [C, D, F, G] [3 3 0 0 1 1]
[Fsus2/E] [C, E, F, G] [x 3 3 0 1 0]
[Fsus4/G] [A#, C, F, G] [x 3 5 3 6 3]
[G, GM, Gmaj] [B, D, G] [x 10 12 12 12 10]
[G#5, Ab5] [B, D#, G] [3 2 1 0 0 3]
[G/A] [A, B, D, G] [3 0 0 0 0 3]
[G/C] [B, C, D, G] [3 3 0 0 0 3]
[G/E] [B, D, E, G] [0 2 0 0 0 0]
[G/F] [B, D, F, G] [1 x 0 0 0 3]
[G/Gb] [B, D, F#, G] [2 2 0 0 0 3]
[G5] [D, G] [3 5 5 x x 3]
[G6] [B, D, E, G] [0 2 0 0 0 0]
[G6/add9] [A, B, D, E, G] [0 0 0 0 0 0]
[G7] [B, D, F, G] [1 x 0 0 0 3]
[G7/add11] [B, C, D, F, G] [x 3 0 0 0 1]
[G7sus4] [C, D, F, G] [3 3 0 0 1 1]
[G9] [A, B, D, F, G] [x 0 0 0 0 1]
[Gadd9] [A, B, D, G] [3 0 0 0 0 3]
[Gaug/E] [B, D#, E, G] [3 x 1 0 0 0]
[Gb, F#, GbM, Gbmaj, F#M, F#maj] [A#, C#, F#] [2 4 4 3 2 2]
[Gb#5] [A#, D, F#] [x x 0 3 3 2]
[Gb/Ab, F#/Ab] [A#, C#, F#, G#] [x x 4 3 2 4]
[Gb/E, F#/E] [A#, C#, E, F#] [2 4 2 3 2 2]
[Gb/Eb, F#/Eb] [A#, C#, D#, F#] [x x 1 3 2 2]
[Gb/F, F#/F] [A#, C#, F, F#] [x x 3 3 2 2]
[Gb6, F#6] [A#, C#, D#, F#] [x x 1 3 2 2]
[Gb7, F#7] [A#, C#, E, F#] [2 4 2 3 2 2]
[Gb7(#5), F#7(#5)] [A#, D, E, F#] [2 x 4 3 3 0]
[Gb7/#9, F#7/#9] [A, A#, C#, E, F#] [x 0 4 3 2 0]
[Gb7sus4, F#7sus4] [B, C#, E, F#] [x 4 4 4 x 0]
[Gbadd9, F#add9] [A#, C#, F#, G#] [x x 4 3 2 4]
[Gbaug/E, F#aug/E] [A#, D, E, F#] [2 x 4 3 3 0]
[Gbdim/D, F#dim/D] [A, C, D, F#] [x 0 0 2 1 2]
[Gbdim/E, F#dim/E] [A, C, E, F#] [x 0 2 2 1 2]
[Gbdim/Eb, F#dim/Eb] [A, C, D#, F#] [x x 1 2 1 2]
[Gbdim7, F#dim7] [A, C, D#, F#] [x x 1 2 1 2]
[Gbm, F#m, Gbmin, F#min] [A, C#, F#] [2 4 4 2 2 2]
[Gbm/D, F#m/D] [A, C#, D, F#] [x x 0 14 14 14]
[Gbm/E, F#m/E] [A, C#, E, F#] [0 0 2 2 2 2]
[Gbm7, F#m7] [A, C#, E, F#] [0 0 2 2 2 2]
[Gbm7(b5), F#m7(b5)] [A, C, E, F#] [x 0 2 2 1 2]
[Gbm7/b9, F#m7/b9] [A, C#, E, F#, G] [0 0 2 0 2 2]
[Gbmaj7, F#maj7, GbM7, F#M7] [A#, C#, F, F#] [x x 3 3 2 2]
[Gbsus, F#sus] [B, C#, F#] [x 4 4 4 2 2]
[Gbsus2/Bb, F#sus2/Bb] [A#, C#, F#, G#] [x x 4 3 2 4]
[Gbsus4/E, F#sus4/E] [B, C#, E, F#] [x 4 4 4 x 0]
[Gdim/E] [A#, C#, E, G] [x 1 2 0 2 0]
[Gdim/Eb] [A#, C#, D#, G] [x 1 1 3 2 3]
[Gdim7] [A#, C#, E, G] [x 1 2 0 2 0]
[Gm, Gmin] [A, C, E] [5 7 7 5 5 5]
[Gm/E] [A#, D, E, G] [3 x 0 3 3 0]
[Gm/Eb] [A#, D, D#, G] [x 6 8 7 8 6]
[Gm/F] [A#, D, F, G] [3 5 3 3 3 3]
[Gm13] [A, A#, D, E, F, G] [0 0 3 3 3 3]
[Gm6] [A#, D, E, G] [3 x 0 3 3 0]
[Gm7] [A#, D, F, G] [3 5 3 3 3 3]
[Gm7/add11] [A#, C, D, F, G] [x 3 3 3 3 3]
[Gm9] [A, A#, D, F, G] [3 5 3 3 3 5]
[Gmaj7, GM7] [B, D, F#, G] [2 2 0 0 0 3]
[Gsus] [C, D, G] [x 3 0 0 3 3]
[Gsus2] [A, D, G] [5 x 0 0 3 5]
[Gsus2/B] [A, B, D, G] [3 0 0 0 0 3]
[Gsus2/C] [A, C, D, G] [x x 0 2 1 3]
[Gsus2/E] [A, D, E, G] [x 0 2 0 3 0]
[Gsus2/Gb] [A, D, F#, G] [5 x 4 0 3 5]
[Gsus4/A] [A, C, D, G] [x x 0 2 1 3]
[Gsus4/B] [B, C, D, G] [3 3 0 0 0 3]
[Gsus4/E] [C, D, E, G] [3 x 0 0 1 0]
[Gsus4/F] [C, D, F, G] [3 3 0 0 1 1]

B

psscatter.m

```
function s = psscatter(X,Y,PointSize,DataPointLocation)

% psscatter(X,Y,PointSize,DataPointLocation) creates a scatter plot of X and Y in Adobe Photoshop.
% PointSize (optional) sets the pixel width of the data point.
% DataPointLocation (optional) sets the data point to resemble an image at the given path.

if nargin < 2
    error('psscatter requires two input variables, X and Y.')
end
if length(X) $\sim$= length(Y)
    error('X and Y are not the same length.')
end
if nargin < 3
    PointSize = 5;
end
datapoint = 0;
pslaunch;
if nargin < 4
    datapoint = [5,156,25,15,5;15,30,50,30,15;25,50,65,50,25;15,30,50,30,15;5,15,25,15,5];
    datapoint = datapoint;
else
    psopendoc(DataPointLocation);
    datapoint = psgetpixels();
end
ImgSize = max(max(X),max(Y));
OldSize = size(datapoint);
OldSize = OldSize(1);
datapoint = uint8(round(imresize(datapoint,PointSize/OldSize)));
X = round((ImgSize/max(X))*X);
Y = round((ImgSize/max(Y))*Y);
psconfig('pixels', 'pixels', 10, 'no');
psnewdoc(ImgSize,ImgSize,72,'Scatter Graph','rgb','backgroundcolor',1.0,8);
p = psgetpixels();
p(p>0) = 0;
pssetpixels(p);
```

```
p = psgetpixels();
l = size(Y);
l = l(1);
for i = 1:l
    sy = 1;
    sx = 1;
    xl = PointSize-1;
    yl = PointSize-1;
    y = ImgSize-Y(i,1)-round(PointSize/2);
    x = X(i,1)-round(PointSize/2);
    if (x < 1)
        xl = xl-(1-x);
        x = 1;
    end
    if (y < 1)
        yl = yl-(1-y);
        y = 1;
    end
    if (x > ImgSize-PointSize)
        xl = round(PointSize/2);
        sx = round(PointSize/2)-1;
        x = ImgSize-PointSize;
    end
    if (y > ImgSize-PointSize)
        yl = round(PointSize/2);
        sy = round(PointSize/2)-1;
        y = ImgSize-PointSize;
    end
    if (x+xl > ImgSize-1)
        xl=ImgSize-1-x;
    end
    if (y+yl > ImgSize-1)
        yl=ImgSize-1-y;
    end
    p(y:(y+yl),x:(x+xl),1) = p(y:(y+yl),x:(x+xl),1)+datapoint(sy:(sy+yl),sx:(sx+xl));
    p(y:(y+yl),x:(x+xl),2) = p(y:(y+yl),x:(x+xl),2)+datapoint(sy:(sy+yl),sx:(sx+xl));
    p(y:(y+yl),x:(x+xl),3) = p(y:(y+yl),x:(x+xl),3)+datapoint(sy:(sy+yl),sx:(sx+xl));
end
s = size(p);
for x = 1:s(1)
    for y = 1:s(2)
        for l = 1:s(3)
            p(x,y,l) = 255-p(x,y,l);
        end
    end
end
pssetpixels(p);
```