Winter 2010-11-01

A Similarity Matrix for Irish Traditional Dance Music

Padraic Lavin
Technological University Dublin, padraic.lavin@student.dit.ie

A similarity matrix for Irish traditional dance music

Padraic Lavin

A dissertation submitted in partial fulfilment of the requirements of
Dublin Institute of Technology for the degree of
M.Sc. in Computing (Information Technology)

July 2010

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Information Technology), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research.

Signed: _____

Date: ***26 July 2010***

ABSTRACT

It is estimated that there are between seven and ten thousand Irish traditional dance tunes in existence. As Irish musicians travelled the world they carried their repertoire in their memories and rarely recorded these pieces in writing. When the music was passed down from generation to generation by ear the names of these pieces of music and the melodies themselves were forgotten or changed over time. This has led to problems for musicians and archivists when identifying the names of traditional Irish tunes.

Almost all of this music is now available in ABC notation from online collections. An ABC file is a text file containing a transcription of one or more melodies, the tune title, musical key, time signature and other relevant details.

The principal aim of this project is to define a process by which Irish music can be compared using string distance algorithms. An online survey will then be conducted to assess if human participants agree with the computer comparisons. Improvements will then be made to the string distance algorithms by considering music theory. Two other methods of assessing musical similarity, Breandán Breathnach's Melodic Indexing System and Parsons Code will be computerised and integrated into a Combined Ranking System (CRS). An hypothesis will be formed based on the results and experiences of creating this system. This hypothesis will be tested on humans and if successful, used to achieve the final aim of the project, to construct a similarity matrix.

Key words: *Irish music, string distance algorithm, similarity matrix, combined ranking system, music comparison, edit distance*

ADMHÁLACHA

Ba mhaith liom mo bhuíochas a chur in iúl do mo mhaor, an Dr. Pierpaolo Dondio, mar gheall ar a fhoighne a inspioráid agus a spreagadh le linn an tionscnaimh seo.

Táim faoi chomaoin mhór ag an Dr. Bryan Duggan a spreag chun an fheachtais seo mé lena shaothar féin agus gan a chabhair agus a achmainn bheadh críoch fhoghanta an tionscnaimh seo uireasach.

Ba mhaith liom mo bhuíochas a ghabháil le Brendan Tierney, Dr. Ronan Fitzpatrick, Dr. Svetlana Hensman, Deirdre Lawless, Paul Doyle agus an fhoireann uilig san Scoil Ríomhaireachta, DIT, Sráid Chaoimhín.

Buíochas do m'fhostóir, An Roinn Talamhaíochta, Iascaigh agus Bia, go háirithe mo bhainisteoirí líne, a bhí is atá agam, as a solúbthacht agus a bhfoighne le seacht mbliain anuas. Ba mhaith liom, freisin, buíochas a ghabháil le foireann oibrí an Aonaid Oiliúna as a gcabhair agus a dtacaíocht.

Ba mhaith liom mo bhuíochas a ghabháil le mo chairde, ceolmhar agus neamcheolmhar, a rinneadh saineolaithe agus neamshaineolaithe díobh mar aidhm an tsuirbhé.

Do mo chailín, Deirdre, is mian liom mo bhuíochas dílis a chur in iúl as ucht a tuiscint, a foighne agus a foinn.

Táim de shíor faoi chomaoin ag mo thuismitheoirí, Pádhraic agus Treasa, mo dheartháir Éanna agus mo dheirféar Treasa as a ngrá agus a dtacaíocht bhuanseasmhach.

Mar fhocal scoir, ba mhaith liom mo bhuíochas ó chroí a ghabháil le mo sheanathair, Philip Lavin, nach maireann, a mhúin dom mo chéad cheol traidisiúnta ó aois a sé agus fós a spreagann mo cheol inniu.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor Dr. Pierpaolo Dondio for his patience, inspiration and encouragement throughout this project.

I am very grateful to Dr. Bryan Duggan whose work inspired this project and without whose help and resources this project would not have been successfully completed.

I would like to thank Brendan Tierney, Dr. Ronan Fitzpatrick, Dr. Svetlana Hensman, Deirdre Lawless, Paul Doyle and all of the staff in the School of Computing, DIT, Kevin Street.

I would like to thank my employer, the Department of Agriculture, Fisheries and Food, in particular my line managers, both past and present for their flexibility and patience over the last seven years. I would also like to thank the staff working in the Training Unit for their assistance and support.

I would also like to thank my musical and non-musical friends who became Irish traditional music experts and non-experts for the purposes of the survey.

To my girlfriend Deirdre, I wish to express my deepest gratitude for her understanding, patience and tunes.

I am forever indebted to my parents, Paddy and Terry, my brother Enda and sister Treasa for their unwavering love and support.

Finally, I would like to express my profound appreciation to my late grandfather, Philip Lavin, who taught me to play Irish traditional music from the age of six and who continues to inspire my music today.

Dedicated to the memory of my grandfather,

Philip Lavin

TABLE OF CONTENTS

Table of Contents

TABLE OF FIGURES

TABLE OF TABLES

LIST OF ABBREVIATIONS

ABC	ABC Notation
C#	C Sharp
CD	Compact Disc
CRS	Combined Ranking System
IDE	Integrated Development Environment
MIC	Melodic Index Code
MIDI	Musical Instrument Digital Interface
MIR	Music Information Retrieval
MP3	Moving Picture Experts Group Layer 3
MS SQL	Microsoft SQL Server
MSM	Music Similarity Matrix
SQL	Structured Query Language

1. INTRODUCTION

"*Music - The one incorporeal entrance into the higher world of knowledge which comprehends mankind but which mankind cannot comprehend.*"

Ludwig van Beethoven (Forbes 1992, p.465)

"*A musician can do no better than pass it on*."

Philip Lavin, 1977

The purpose of this chapter is to provide an introduction to this dissertation. Section 1.1 outlines the project area while Section 1.2 provides a background to Irish traditional dance music. The research problem is presented in Section 1.3 and Section 1.4 explains the intellectual challenge. The research objectives and methodology are summarised in Sections 1.5 and 1.6 respectively. Section 1.7 outlines the resources needed in order to complete this dissertation and its scope and limitations are described in Section 1.8. Section 1.9 concludes with a description of how this dissertation is organised.

1.1 Overview of project area

It is estimated that there are between seven and ten thousand Irish traditional dance tunes in existence (Duggan 2009, p.ii). As Irish musicians travelled the world they carried their repertoire in their memories and rarely recorded these pieces in writing. When the music was passed down from generation to generation by ear the names of these pieces of music and the melodies themselves were changed or forgotten over time.

Most of this music is now available in ABC notation (Walshaw 1995). An ABC file is a text file with an .abc extension containing such details as the tune title, a transcription of one or more melodies, musical key and time signature. For the first phase of the project a corpus of tunes will be analysed using string distance algorithms and used to form a similarity matrix identifying the relationships between different tune parts.

Some programming will be required in order to pre-process the databases of ABC format tunes in order to ensure that reliable data is used. Further programming will be required in order to process the databases using string distance algorithms and to form a similarity matrix.

In the second phase of the project, quantitative research will be performed by testing the results of string distance comparisons on humans, some who have little or no knowledge of Irish music and some who are considered experts.

An hypothesis will then be formed based on experiences and results from the first two phases of the project. Based on these results an improved process will be defined and tested on humans before using this process to construct a similarity matrix.

1.2 Background to Irish traditional dance music

The author has over thirty years experience playing Irish traditional music on tin whistle, concert flute and uilleann pipes. The author also has an interest in computing and computer programming. This project gave the author an opportunity to combine both of these interests in order to analyse the relationships between Irish dance tunes.

Traditional Irish dance music is the native folk music of Ireland. It is played on instruments such as harp, tin whistle, flute, fiddle, uilleann pipes, button accordion, concertina, banjo, piano and harmonica. Bones, bodhrán and spoons are percussion instruments commonly used to accompany the music. Customarily, traditional Irish music was played at Céili dances, at weddings, in village houses and other celebrations in order to accompany dancers. In modern times, it is common for musicians to play Irish music in public houses without dancers for their own entertainment or for the entertainment of others.

1.2.1 Types of Irish traditional dance tune

Several thousand pieces of music called “tunes” comprise the corpus of Irish traditional music. There are a number of types of dance tune including reels, single jigs, slip jigs, slides, polkas, hornpipes, waltzes, schottische’s, strathspey’s and barndances. Each of these types have a different rhythm to suit the dance - reels,

hornpipes, schottische's, strathspey's and barndances are in either 2/4 or 4/4 time signatures. Polkas are in 2/4, waltzes in 3/4, single, double and treble jigs are in 6/8, slip jigs in 9/8 and slides in 12/8. A time signature refers to the number of notes per beat and the length of those notes. A reel in 4/4 has four quarter notes per beat, a polka in 2/4 has two quarter notes per beat, a jig in 6/8 has six one eighth notes per beat while slip jigs in 9/8 and slides in 12/8 have nine and twelve one eighth notes per beat respectively.

1.2.2 Musical keys in Irish traditional music

Irish music is usually played in a variety of musical keys, limited only by a particular instrument. For example, a standard non-keyed uilleann pipe chanter or keyless flute is not fully chromatic and does not have the full range of notes a fiddle or piano would have. This means that tunes in certain keys are difficult (but not impossible) to play on certain instruments and this led to the adoption of modal scales such as dorian and mixolydian into Irish traditional music. The following table represents a non-exhaustive list of common keys played on concert pitch instruments such as tin whistle, flute and uilleann pipes in Irish traditional music;

Table 1: Musical keys common in Irish music (Larsen 2003, p.25)

D Major (Ionian)
G Major (Ionian)
A Major (Ionian)
D Mixolydian
G Mixolydian
A Mixolydian
E Dorian
A Dorian
B Dorian
E Minor (Aeolian)
A Minor (Aeolian)
B Minor (Aeolian)

1.2.3 Tune Structure

All types of traditional Irish dance tune consist of parts that are usually repeated. A simple reel or a jig would usually have a "first" or "low" part (containing notes mostly in the lower octave) and a "second" or "high" part (containing notes mostly in the

higher octave). This is not always true as there are tunes that are played "single" where their parts are not repeated and also some tunes consisting of seven or more parts. It is common to refer to "first" or "second" parts as parts A and B respectively and this is the notation used throughout this dissertation. In a two part tune it is normal to play part A twice followed by part B twice and repeat this pattern a number of times. This table shows how some tunes are commonly constructed.

Table 2: Some example of tune parts and repetition patterns

Tune name	**Part Repetition Pattern**
Morrison's Jig	AABB AABB AABB
The Boys of the Lough Reel	AABB AABB AABB
The Lark in the Morning Jig	ABCD ABCD
The Musical Priest Reel	ABC ABC
The Gold Ring Jig	ABCDEF ABCDEF
The Glass of Beer	AB AB AB

It is normal for tunes to be played in "sets". For example, a jig would normally be followed by one or more jigs that are also repeated appropriately. Table 3 is an example of how a set of jigs might be repeated;

Table 3: A set of jigs

Morrison's Jig	AABB x 3
The Lark in the Morning	ABCD x 2
The Leitrim Fancy	AABB x 3

Before the introduction of radio, television and satisfactory public transport infrastructure in Ireland, Irish music was regionalised with unique styles developing over time in various areas of Ireland. For example, County Donegal in the North West of Ireland is associated with fiddle music, Sliabh Luachra in the South West is associated with polkas and slides and North Connaught in Western Ireland is associated with a particular style of flute playing.

1.2.4 Traditional Music Collections

For the most part of this and the last century the majority of Irish musicians could not read staff notation and they learned the majority of their music by ear. Throughout history a number of respected collectors have catalogued Irish music in order to preserve it.

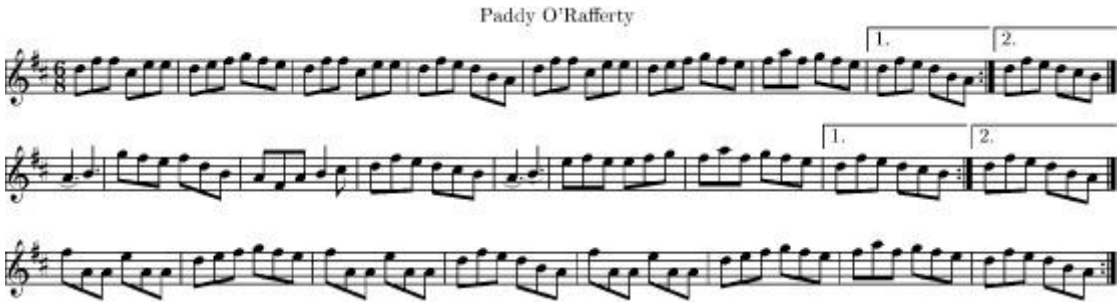


Figure 1: An Irish jig called Paddy O'Rafferty recorded in staff notation

Edward Bunting (1773-1843) collected and published three collections - *A General Collection of the Ancient Irish Music, 66 tunes*, (1796), *A General Collection of the Ancient Music of Ireland* (1809) and *The Ancient Music of Ireland, 165 airs*, (1840) (Bunting 1969). In 1855 George Petrie published *The Petrie Collection of the Ancient Music of Ireland* (Petrie 2002).

Captain Francis O'Neill a policeman living in Chicago, USA published four collections *O'Neill's Music of Ireland* in 1903 containing 1,850 tunes (C. F. O'Neill 1979), *The Dance Music of Ireland* in 1907 containing 1001 tunes (F. O. &. J. O'Neill 1995), *400 tunes arranged for piano and violin* in 1915 and finally *Waifs and Strays of Gaelic Melody* in 1922 containing 365 tunes (O'Neill 1980).

Brendán Breathnach collected more than 7,000 tunes in his lifetime and published five collections entitled *Ceol Rince na hÉireann Cuid I* in 1963 (Breandán Breathnach 1963), *Ceol Rince na hÉireann Cuid II* in 1976 (Breandán Breathnach 1976), *Ceol Rince na hÉireann Cuid III* in 1985 (Breandán Breathnach 1985), *Ceol Rince na hÉireann Cuid IV* in 1996 (Breandán Breathnach 1996), *Ceol Rince na hÉireann Cuid V* in 1999 (Breandán Breathnach 1999). Brendán Breathnach's collections are discussed further in Section 2.2.

1.2.5 Electronic Collections

A number of collectors such as Bill Black (Black 2010), Henrik Norbeck (Norbeck 1996) and Nigel Gatherer (Gatherer 2009) have transcribed traditional collections and their own collections into electronic formats and made them freely available online. ABC notation is now preferred over midi as ABC is text based and can be sight read easily by musicians. Unlike staff notation, ABC notation is already in an electronic format and may be processed by computer systems without the need to convert it into another format.

Websites such as http://www.thesession.org (Keith 2010) facilitates the archiving of tunes by allowing its members submit their tune transcriptions to its database. The tunes available from this collection are of varying quality because they are user submitted and do not always comply with the ABC notation specification. The Session.org website hosts over 9,340 tunes as of April 2010.

A more detailed list of freely available electronic collections of Irish traditional music is available in Section 1.7.4

1.3 Research problem

The principal aim of this project was to evaluate and improve string distance algorithms for the purpose of identifying similarities in the corpus of Irish traditional music. A secondary aim of this project is to define a process by which a Music Similarity Matrix for Irish traditional dance music could be constructed.

Since the 1840's when Ireland was stricken by famine, its people emigrated to England, Europe and the Americas bringing their music, dance and culture with them. The Irish diaspora handed down the music as they inherited it, aurally. Because the music was usually stored in the memory of the musician this led to a number of problems;

- The names of tunes were sometimes forgotten or changed.
- The melodies of tunes were sometimes forgotten or changed.
- Irish music teachers did not always recall tune melodies correctly.
- Students did not always learn the tune exactly as it was taught to them.

As a result, some tunes have multiple names; some tunes have different versions of the melody or completely different melodies. Others share some of the same parts or have phrases that are common in other tunes. Brendán Breathnach, a highly respected collector of Irish music recognised that while he was collecting tunes from any given musician, he could have collected it previously under a different name or that its melody could be similar to another he had collected earlier. The following quotation from Breathnach's first published collection, *Ceol Rince na hÉireann Cuid I,* describes the problem quite well. It lists one traditional Irish tune, "*The Little Yellow Boy*", also known as "*Galloway Tom*", that shares its name with two Scottish tunes with different melodies. The melody of "*The Little Yellow Boy*" or a version of it appears in three Irish tune collections under ten different names;

> *"27. An Buachaillín Buí [The Little Yellow Boy[4]]: I took the name from the version published by O'Farrell in the "Collection of National Irish Music for the Union Pipes" (c.1797). He has two versions in the "Pocket Companion". O'Farrell also called this air Galloway Tom, but if he did it has no relation to the Gallua Tom in the Straloch manuscript or with the Galloway Tom in the "Scots Musical Museum" (325). O'Neill has six versions in the "Music of Ireland", four of them unknown to himself, one would think: The Little Yellow Boy (706); Galway Tom (744/5); The Thrush's Nest (855); The Goat's Horn (926); and The Spotted Cow (983). He has two settings in the "Dance Music of Ireland", Galway Tom (34) and The Spotted Cow (199). Joyce calls it Galway*

Tom (J ii, 806). Nowadays it is usually called The Lark in the Morning, but it is also called Come in the Evening, The Kelso Races, The Welcome and A Western Lilt."

(Breandán Breathnach 1963)

1.4 Intellectual challenge

The first challenge was to obtain clean ABC data, pre-process and store it in a state that was suitable for conducting string comparison experiments. Because some online databases contained user contributed ABC notation it was not always correctly transcribed or did not comply with ABC notation rules. Unreliable data had to be identified automatically and discarded leaving only clean, validated data in the electronic corpus.

The author had access to about twelve thousand five hundred tunes in ABC format with each of these tunes having at least two parts. If each part was to be compared with each other part in the entire corpus this means that there would be n(n-1) comparisons where n is the number of parts in the corpus. A corpus of tunes having 12,500 tunes with at least two parts each would therefore result in 624,975,000 comparisons (25,000 x 24999). The second challenge was to implement algorithms in Java and to design large, efficient databases capable of storing millions of results that could be queried at will using Structured Query Language (SQL) (Chamberlin & Boyce 1974).

The third challenge was to identify existing string distance algorithms that could potentially be used or adapted in order to identify similarities between strings of ABC notes. This involved an investigation of the features, advantages and disadvantages of numerous string distance algorithms and then assessing if they possessed any qualities that could be adapted and applied from a music theory perspective.

The fourth challenge was to present the results in a meaningful way.

An overall intellectual challenge is to show that computer algorithms can be used to find similarities between melodies in Irish traditional music.

1.5 Research objectives

The following objectives have been achieved throughout the dissertation and contributed to the overall outcome:

- To identify and evaluate suitable string distance algorithms for the purpose of conducting comparisons between sequences of musical notes.
- To improve suitable string distance algorithms by implementing features unique to musical theory.
- To test what is meant by a similarity in the context of traditional Irish music. The author felt that the people best positioned to decide similarities in Irish music are the musicians playing Irish music. A survey of both accomplished musicians and non-musicians was conducted in order to validate or disprove the results of computerised experiments.
- To construct a Music Similarity Matrix (MSM) for the corpus of Irish traditional dance music.

1.6 Research methodology

The research methodology used during the project is described in this section. Both primary and secondary research was conducted throughout the duration of the dissertation. The secondary research consisted primarily of the following;

- Identifying collections of Irish traditional tunes in ABC notation that were suitable for computerised comparison.
- Literature review of
    - Online ABC databases
    - Integrated Development Environments (IDE's)
    - Java and C Sharp programming discussion forums
    - Journals
    - Articles
    - White papers
    - Various string distance algorithms
    - Emails from world experts

- Interviews with a world expert, Dr. Bryan Duggan

The primary research consisted of the following;

- Conducting computerised experiments in order to compare ABC tune parts using five string distance algorithms – Levenshtein (Levenshtein 1966), Jaro-Winkler (W. E Winkler 1999), Semex (K Lemström & Perttu 2000) and two new algorithms based on Parsons Code (Parsons 1975) and the Melodic Indexing System developed by Breandán Breathnach (Brendan Breathnach 1982).
- Conducting two online surveys of experts and non-experts in Irish traditional music to test if humans felt that computer selected pairs of tune parts were similar or different.
- Conducting quantitative analysis of the survey.

By conducting experiments on transcribed tunes in ABC notation, a process was formulated whereby computer algorithms could be used to identify similarities between Irish traditional music tune parts. The process was further refined by altering the string distance algorithms in order to take account of features unique to Irish music and an hypothesis was formed. In order to prove or disprove this hypothesis the results were tested on experts and non-experts in the field of Irish traditional music.

The following four phases were planned and carried out in order to complete the project successfully;

1.6.1 Phase one – Collection of tunes in ABC notation

A number of ABC collections exist online and these are described in greater detail in Section 1.7.4. These ABC files were processed automatically in order to separate them into tune parts and imported into a relational database for further processing.

1.6.2 Phase two - Conduct programming experiments

Phase two involved the evaluation of various Integrated Development Environments (IDE's) and short-listing them. This phase also involved the evaluation of string

distance algorithms suitable for music comparison and implementing them within a framework for conducting music comparison experiments.

1.6.3 Phase three – Survey of experts and non-experts

After tune pairs had been selected using Breathnach's Melodic Indexing System and the Levenshtein (Levenshtein 1966) and Jaro-Winkler (W. E Winkler 1999) algorithms, the original ABC tune pairs were converted from ABC text notation to mp3 audio files and included in an online survey. Expert and non-expert participants were invited to complete the survey and their choices were recorded.

1.6.4 Phase four - Conclusions drawn from analysis of survey

In order to evaluate the hypothesis it was necessary to analyse how the computer selected tune pairs were viewed by experts and non-experts completing the survey. Because music similarity can be very subjective, careful and empirical analysis of the results was necessary.

1.6.5 Phase five – Construction of a Similarity Matrix

Once the analysis in phase four was completed a process was designed whereby strings of musical notes could be compared by combining scores from multiple algorithms. This process was tested on humans in a second online survey and then used to construct the similarity matrix for Irish traditional music.

1.7 Resources

1.7.1 Library Facilities

An extensive literature review was carried out in order to complete this project. A number of world experts have published relevant articles on music comparison and their knowledge contributed greatly to the success of this project.

1.7.2 Programming Environment and Database Server

Various Integrated Development Environments (IDE's) were obtained and a shortlist of possible solutions was created;

- Microsoft Visual Studio 2008 Professional Edition and Microsoft SQL Server 2008 Developer Edition. Both applications are available to eligible students at no charge through the Microsoft Dreamspark program (Microsoft Corp. 2010).
- Eclipse Java IDE with MySql Database Server, also free of charge.
- Netbeans Java IDE with the integrated Derby database server, also available free of charge.

Netbeans and Derby (Sun Microsystems 2010) were chosen over the other two solutions in order to complete the first four phases for three main reasons;

- Java implementations of the Levenshtein, Jaro-Winkler and Lemström algorithms were available and this would have the effect of reducing the amount of development, testing and debugging time if a Java IDE were used (Microsoft Visual Studio 2008 does not support Java).
- Having a database server integrated within the IDE meant that a complete solution would be in place after one simple install without the need to install or configure a separate database server.
- Familiarity with Netbeans meant that less time would be spent learning how to use the development environment leaving more time for designing, programming and running experiments.

Because of performance problems with the Netbeans / Derby platform, the final phase of the project (the completion of the similarity matrix) was completed using Microsoft Visual Studio 2008 and Microsoft SQL Server 2008 Developer Edition. Moving to this platform also allowed the author to harness the power and simplicity of using custom Structured Query Language (SQL) functions to perform string distance comparisons.

1.7.3 Access to a supervisor

Weekly meetings with a supervisor were a necessary resource for the successful completion of this project. The supervisor assigned to this project was Dr. Pierpaolo Dondio whose insightful guidance and advice contributed immensely to the successful completion of this project.

1.7.4 Providers of databases of Irish tunes in ABC Notation

- The Irish Traditional Music Archive
- The Session.org (Keith 2010)
- Henrik Norbeck (Norbeck 1996)
- O'Neills Music of Ireland ("1850"), Dance Music of Ireland ("1001") and Waifs and Strays of Gaelic Melody (Chambers 2010b)
- Ceol Rince na hÉireann Cuid I, II, III, IV (Black 2010)
- Johnny O'Leary of Sliabh Luachra (Black 2010)
- Nigel Gatherers ABC Collection (Gatherer 2009)
- John Chambers Tune Finder (Chambers 2010a)

1.7.5 Two groups of survey participants

In order to test if computer selected traditional Irish tune parts sound similar, a survey of non-experts and experts in the field of Irish music were surveyed and their responses recorded.

1.8 Scope and limitations

The source ABC data contains melody, time signature, musical key, tune title and other pertinent information. No information on playing style exists in the ABC files. This project will therefore be limited to assessing similarity based primarily on melody.

A large percentage of ABC files used as the source data for this project were transcribed by humans of differing musical ability and did not conform absolutely to the ABC notation specification. Resource constraints limited the amount of data that could be corrected manually and as a result most of the problematic data was discarded as it was unreliable.

The corpus of Irish traditional music contains "exact melody matches" where the names are different but the melodies identical. It also contains "exact name matches" where the dance tunes have identical names but different melodies. ABC notation already supports multiple tune titles in its specification (Walshaw 1995). Although exact name and melody matches would form part of a music similarity matrix this aspect of the matrix was not focussed on as identifying them does not present a significant challenge.

Musical similarity in the context of this project would have the following characteristics;

- Pairs of tune parts where the melodies are not exact matches.
- Sequences of musical notes that contain common phrases or sub-sequences.
- Where a musical similarity can be expressed as a value between 0 and 1.

Finally, this project is not concerned with how humans perceive melodic similarity merely that humans can compare and identify instances where music sounds alike.

1.9 Organisation of the dissertation

This dissertation comprises of an introduction and six other chapters as follows;

- Chapter 2 explores the meaning of music comparison and how researchers and music collectors have defined systems in order to measure or express how similar or different two musical pieces are. The work of the renowned music collector and world expert, Breandán Breathnach is described along with the system he devised in order to prevent duplicate Irish traditional dance tunes from entering his collections. This chapter concludes with a brief introduction to ABC notation (a specification for transcribing music in text format) and an overview of why ABC was chosen for use in this project.
- Chapter 3 begins by explaining what a string distance algorithm is and continues by defining what "similarity" or "dissimilarity" means in the context of music and in particular, Irish traditional dance music. Some music theory considerations are also presented along with theories of how these concepts might be implemented within a string distance algorithm. The Levenshtein, Jaro-Winkler and Lemström Semex algorithms are then explained in detail with examples of how each might be used.
- Chapter 4 outlines three contributions to the body of knowledge in the string distance domain. The first contribution concerns the weighting of melodic sequence variations and how this technique can be used to identify two very similar pieces of music that would not otherwise have been identified. The

second contribution relates to the weighting of short note prefixes that sometimes precede Irish tunes. Breandán Breathnach, a prominent collector of Irish traditional music recognised the importance the start of a musical piece has in relation to identifying a tune while William E. Winkler, an academic working for the US Census Bureau recognised the extra significance that matching a rarely occurring item had in correctly matching two records. How these techniques could be applied to Irish music is then examined. Possible improvements to the Levenshtein algorithm are also presented and some conclusions drawn. Contribution three, a method of using ranking to assess the accuracy of a similarity match is then illustrated.

- Chapter 5 presents the advantages of Breandán Breathnach's Melodic Indexing System contrasted with some disadvantages and tradeoffs. Some proposed improvements are outlined before contribution 4, the computerisation of the Melodic Indexing System, is presented. Contribution 5 concerns how sorting index codes numerically is not feasible for different length index codes and offers a solution to the problem.
- Chapter 6 presents an overview of the data used for the purposes of performing string distance and music comparison experiments. The design issues that were faced while constructing experiments and surveys are discussed before an overview of each experiment and survey is given. Results of the experiments, surveys and their analysis conclude the chapter.
- Chapter 7 gives an overview of the research domain and describes the research performed during this project. Summaries of the contributions to the body of knowledge are then given. Synopses of the experimentation and evaluation phases are outlined before the scope of the project limitations are discussed. The research objectives achieved are also presented before future work and research areas are identified. Finally, some conclusions are presented before ending the chapter.

2. MUSIC COMPARISON TECHNIQUES

2 Introduction

The purpose of this chapter is to review methods of assessing music comparison other than by using string distance algorithms. Two techniques for assessing music similarity are presented, Breandán Breathnach's Melodic Indexing System (MIC) introduced in the 1960's and Parsons Code, invented by Dyers Parsons in 1975. The question of what exactly a similarity means in the context of Irish traditional music is explored.

2.1 What is music comparison?

In the context of this project, music comparison using string distance algorithms, Parsons Code or Breathnach's MIC means a measure of similarity that can be expressed as a value between 0 and 1 with 0 meaning completely different and 1 meaning an exact match. In all cases, the result of a comparison was normalised so that the results of each method could be compared.

For example, the Levenshtein algorithm returns the number of edits it would take to convert one string into another by using character insertions, deletions and substitutions. Section 3.2 outlines a more detailed explanation of the Levenshtein algorithm. The Levenshtein algorithm could return a result of 0, 1 or any number greater than 1. In order to express this result as a normalised score between 0 and 1 the number of Levenshtein edits was divided by the length of the longest string and subtracted from 1. Comparing two identical strings of notes returns an edit distance of 0 resulting in a normalised value of 1 as 1-(0/string length) = 1. Because the maximum number of edits returned is equal to the length of the longest string, two completely different strings of notes will return a result of 0 as 1-(string length/string length) = 0. Any edit distance between 1 and the length of the longest string will result in a proportionate value between 0 and 1.

By normalising the results of each algorithm so that they all return a result between 0 and 1 this allows for the comparison of the algorithms themselves. It also enables the

rankings of the results of each algorithm and these can be used to generate standard deviation scores for each comparison.

Music comparison of audio recordings is a popular research topic with various techniques being developed by researchers in order to solve diverse problems related to the field of Music Information Retrieval (MIR) such as comparing music using sung queries (Hu & R. B Dannenberg 2002), retrieving music using graph invariants (Pinto & Haus 2007), improving music retrieval by compacting musical signatures (Cui et al. 2008), computing approximate repetitions in musical sequences (Cambouropoulos et al. 2001), creating models of musical similarity by using self-organising maps (Toiviainen & Eerola 2002) and using entropy based fingerprints to identify musical performances (Camarena-Ibarrola & Chávez 2006).

A variety of string distance algorithms have been used to compare pairs of sequences of notes, the most popular of which is the Levenshtein method (Levenshtein 1966). This project uses the Levenshtein algorithm with implementations of the Jaro-Winkler algorithm (W. E Winkler 1999), Lemström and Perttu's Semex algorithm (K Lemström & Perttu 2000), Parsons Code (Parsons 1975) and a new algorithm based on Breandán Breathnach's work, implemented and improved by the author, to perform comparisons on fragments of Irish traditional dance tunes called "parts".

2.2 Brendán Breathnach

Brendán Breathnach (1912-1985) was a respected collector and cataloguer of Irish traditional music. He collected more than 7,000 tunes in his lifetime while working as a civil servant in the Department of Education and after he retired. He is most well known for his five volume collection, Ceol Rince na hÉireann Cuid I, II, III, IV & V (Breandán Breathnach 1963) two editions of which were published after his death.

While editing the first volume of Ceol Rince na hÉireann in 1963, Brendán Breathnach recognised that he may have collected the same tune previously or that it may already be contained in other collections such as Captain Francis O'Neill's *Dance Music of Ireland – 1001 Gems* (F. O'Neill 1907). Wanting to include only previously

unpublished tunes in his collection, he developed an indexing system specifically designed for Irish music similarity detection.

Figure 2: Breandán Breathnach Melodic Indexing System. Source: Author

Breathnach described his indexing system briefly in his article *Between the jigs and the reels* (Brendan Breathnach 1982, pp.43-48). The system was based on the theory that a tune could be identified from the first two bars, commonly referred to in musical terms as an "incipit". Index cards were created for every tune in known collections and for newly collected and transcribed tunes. The index cards contained the following information;

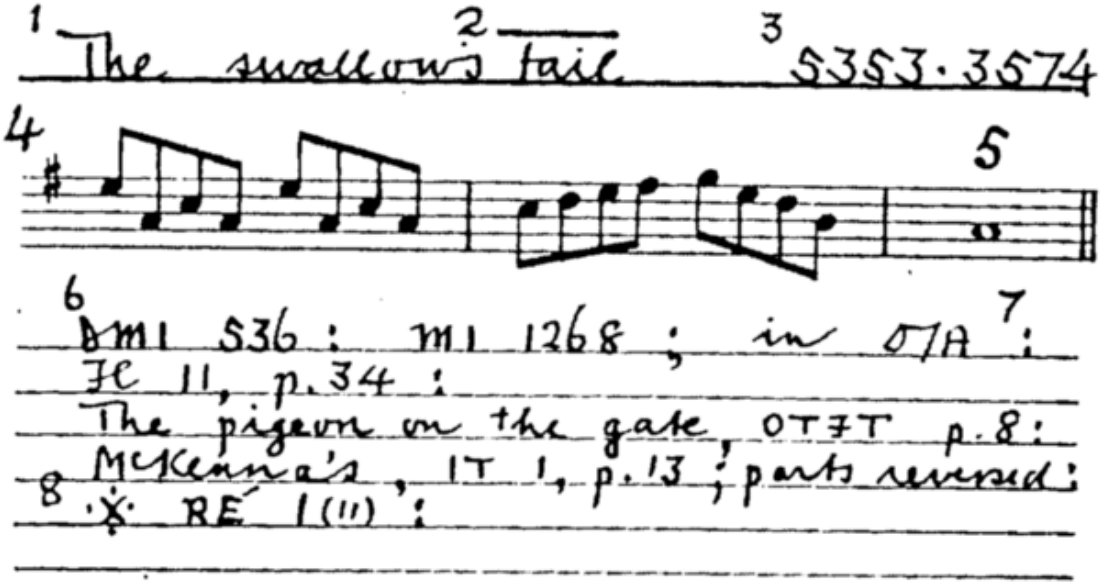


Figure 3: An index card from the Breandán Breathnach Melodic Index for the tune "The Swallows Tail" (Brendan Breathnach 1982)

1. The tune title
2. Numerical series
3. Code generated from the first two bars
4. Staff notation of the first two bars
5. The final note of the tune
6. The source of the tune e.g. published collection
7. Comments
8. Audio recording of the tune

Index cards were created for each tune in published collections and for tunes that were newly collected and transcribed. They were stored sequentially according to the code at 3 in Figure 3 above.

The generation of the code is of particular interest to this project as it is transposition invariant. This means that tunes transcribed in different musical keys may be compared without the need for transposing to a common key. In order to calculate the code the final note must be ascertained. Usually, but not always, the final note of a tune will represent the key in which the tune is played. Using this final note as the tonal centre

of the tune, sequential notes preceding and following it are given values in steps of 1 as in Figure 4 below;

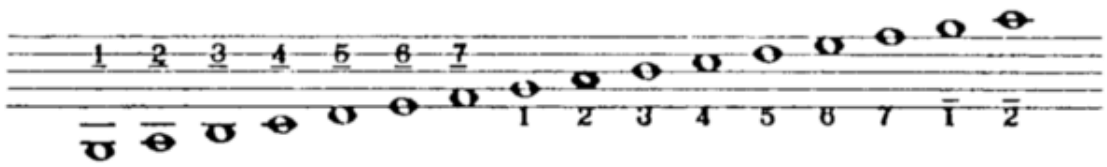


Figure 4: Assigning numerical values from a final note (Brendan Breathnach 1982)

The final note "G" in the centre above is given a value of 1 with notes before and after it calculated appropriately. Notice that the "G" notes an octave above and below the final note appear on the right and left hand sides respectively also have a value of 1. This effectively gives all "G" notes the same value irrespective of the octave they are contained within. In other words, Breathnach is suggesting that low, middle and high "G" notes are equal and that the octave in which a note is played has no bearing on melodic comparison.

Table 4: Note values calculated with a fundamental note of A

Note	A	B	C	D	E	F	G
Value	1	2	3	4	5	6	7

The notes contained in the first two bars of "The Swallows Tail" in Figure 3 are EACA EACA CDEF GEDB. Extracting the accented notes from this phrase yields ECEC CEGD. Accented notes are notes within the phrase that are dominant or stressed and in this case it means that every second note is dominant i.e. notes 1, 3, 5, 7, 9, 11, 13 and 15. Substituting the values for the notes in Table 4 the code 5353 3574 is obtained. This code represents a transposition invariant signature derived from the melody of the tune that can be compared to other tune signatures. By ordering tunes numerically by code, duplicates are identified.

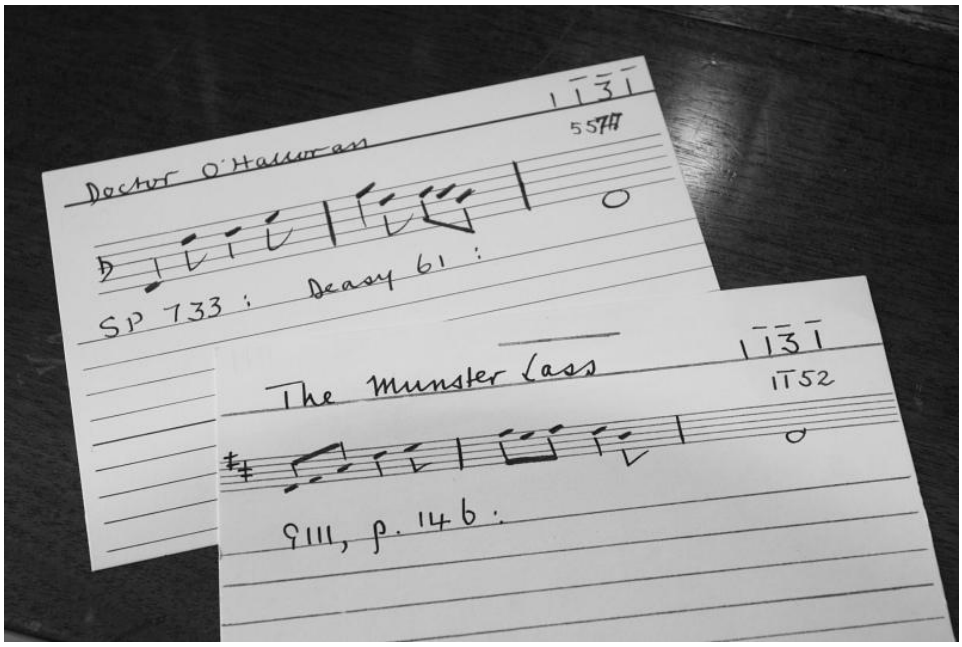


Figure 5: Two jigs from Breandán Breathnach's collection, Doctor O'Halloran and The Munster Lass. Source: Author

Breathnach did not define a method to assign scores based on proximity to a match. A new system was developed in order to return similarity scores comparable to those returned by the string distance algorithms. Normalised scores were calculated by obtaining the distance from the match, dividing it by the number of tunes in the corpus (the maximum distance) and subtracting it from 1. The same method of calculating normalised scores was used for both Melodic Indexing and Parsons Code systems. This process is discussed in greater detail in paragraph 2.3.1.

2.3 Parsons Code

In 1975 Denys Parsons introduced a system of identifying musical pieces by comparing their melodic contour. The system is very simple and very effective. The first note of a piece of music is used as a point of reference and is represented as an asterix. Each subsequent note is given a value of U, D or R depending on if it is higher, lower or equal to the note preceding it.

For example, the musical notes ABCCABDD would be represented as *UURDUUR. Parsons Code also has the advantage of being transposition invariant as comparison is not affected by the musical key of the piece. Like the earlier example, the musical notes BCDDBCEE are also represented by the same Parsons Code *UURDUUR.

This method of music comparison is easily understood by non-musicians and allows people to express a piece of music by contour relatively easily regardless of musical ability and without the need to recognise notes, musical key or time signature.

In order to normalise Parsons Code it was necessary to calculate the Parsons Code for the whole corpus of tune parts and sort them alphabetically (by Parsons Code) as follows;

ID	Name	Notes	Parsons Code	Distance
15351	Colonel Robertson	AFEDAADBGBAAAFEDAADCCEEEAFEDAADI	*DDDURDUDUDRRDDDURDDRURRUDDD	9
9323	Langstrom"s Pony	FEDCAAEAACAAFEDCAABGBDCBFEDCAAEA	*DDDURDURDURDDDDURUDUDDUDDDI	8
11283	Langstrom"s Pony	FEDCAAEAACAAFEDCAABGBDCBFEDCAAEA	*DDDURDURDURDDDDURUDUDDUDDDI	7
18139	Palm Sunday	BAGEGGEGGEGGEAABCCDEEDEFGEEEDBAB	*DDDURDURDURDURUDRUURDUUUDRR	6
9738	Palm Sunday	BAGEGGEGGEGGEAABCCDEEDEFGEEEDBAB	*DDDURDURDURDURUDRUURDUUUDRR	5
18553	Pendle Witches, The	BAFDAADEEAEEFGBEDCCADBGBCDBEEABG	*DDDURDURUDRUUUDDDRUDUDUDUUI	4
8657	Glassan	BAFDFFDEEFGGBAFDFFDEEEBBBAFDFFDEEI	*DDDURDURUURUDDDURDURRURRDDD	3
19092	Roddie C"s	AFECAAACDBCABGGBAFECFBBCECBCAAAA	*DDDURRDUUDUUDRUDDDDUURDUDUI	2
19520	Gan Ainm	FEDCAAAACEAFEFGAAABCAAAFAAGFFFGC	*DDDURRRDUUDDUUURRUDURRDURDD	1
9253	Down the Hill	BAGEAAAAAAEABCBABCABAGABGAGEDEFG	*DDDURRRRDUUDUDUDUUDDUUDUDDI	0
17437	Each Little Thing	BGFEGGGGGBGFEEEBCEGFFFFFGFEBCEEEEF	*DDDURRRRUDDDRRUDUUDRRRRUDDUI	1
16752	Harbourview, The	GFEDEEEFEDEEFBABDCDAFDDDGFEDEEEFEI	*DDDURRUDDURUUDUDDUUDDRRUDDI	2
8848	Biddy Martin	FEDCEEFEDFEDCEEADD	*DDDURUDDUDDDURUDR	3
13601	An Dro	FEDCDDADBAGBAAAGFFFGAABAGABGEEEI	*DDDURUDUDDUDRRDDRRUURUDDUUD	4
17147	Girls Of The Town	BAFEAABCFFECBAFEAABCCCBAFAFEAABCF	*DDDURUDURDDUDDDURUDRRUDDUDI	5
11186	Castle Kelly	AGECDFEDCAGEGGEGAGECDDCDEAAGAAB	*DDDUUDDDUDDURDUUDDDURDUUURI	6
16278	Kathleen"s Wedding	AGFEFAGFEFDDDDDEGACAAAAFFEFDCAF	*DDDUUDDDUDRRRRRUUUDURRRDRDU	7
13404	Jackson"s Morning Brush	GFEDFAFEDEFDBAFADEFGFGEEGFEDFAFED	*DDDUUDDDUUDUDDUDUUUDUDRUDDI	8
16604	North Sydney Bar	AGECGADCAGECEDCACGECGABCAGAGECC	*DDDUUDDUDDDUDDUDUDDUUUDUDU	9

Figure 6: Parsons Code Calculation and Distance

2.3.1 Normalised Parsons Code Scores

Once the corpus has been converted to Parsons Code a match to the search term can be identified. The search term in this case was the Parsons Code of the tune with ID 9253 – "Down the Hill" in Figure 6 above. This exact match is given a distance of 0 with the next closest match in either direction given distances in ascending order. In the case of a closest match (as opposed to an exact match) a distance of 0 from the search term is also given. Figure 6 shows that the tune "Down the Hill", ID 9253 is an exact match and has a distance of 0 from itself. The next closest match in each direction is given a distance of 1 greater than the preceding row i.e. tunes with ID 17437 and 19520 have a distance of 1, tunes with ID 16752 and 19092 have a distance of 2 and so on. This method of ranking rows of results has been termed MICRank for the purposes of this project.

In Figure 6 the search term is the Parsons code for tune ID 9253 i.e. *DDDURRRRDUUDUDU. The tune with ID 19520 is represented in Parsons Code as *DDDURRRDUUDDUUU and the tune with ID 17437 is represented as *DDDURRRRUDDDRRU. The search match is given a distance of 0 and the other two tunes are given a distance of 1.

If these tunes were to be ranked in order of closeness instead of calculating distance they would be ranked as follows;

1. ***DDDURRRR**DUUDUDU – ID 9253
2. ***DDDURRRR**UDDDRRU – ID 19520
3. ***DDDURRR**DUUDDUUU – ID 17437

Note that the tune with ID 19520 is a closer match to the search term as the first 9 notes are identical, compared with the first 8 notes of the tune with ID 17437. This method of ranking rows of results has been termed MICDenseRank.

The same method of calculating normalised scores is used for the computerised versions of the Parsons Code and Breathnach's Melodic Indexing Systems. In the original Melodic Indexing System these two tunes would have been physical index cards either side of the matched tune, each a distance of 1 from it. Similarly, the code used to calculate proximity in this project returns equal distances from the match for these tunes.

This possible inaccuracy in the way distance is calculated in the computerised versions of the Melodic Indexing System and Parsons Code was identified but not changed for two principal reasons;

- The original intention was to mimic the original Melodic Indexing System.
- Increasing the accuracy of the algorithm would negatively affect performance drastically.

Improving the precision of Parsons Code and Melodic Indexing System ranking has been identified as an area for further investigation, future work and development.

In order to normalise distance from a match the following formula was used;

$$1 - (\text{distance} / \text{maximum distance})$$

Therefore an exact match in a corpus of 11944 tunes would have a normalised score of 1, calculated as follows;

$$1 - (0 / 11944)$$

A tune with a distance of 5000 from a match would be calculated as follows;

$$1 - (5000 / 11944)$$

$$1 - (0.418)$$

$$0.582$$

In the final version of the algorithm, normalised scores are calculated for all results in the corpus and ranked in order of score.

2.4 ABC Notation

Traditionally, most western music is written using staff notation which can be sight read by musicians. It consists of symbols that represent notes, rests, repetitions, musical key, time signature and other musical concepts written on a five line staff. As it is image based it does not lend itself to being as easily machine processed as the text based ABC notation.

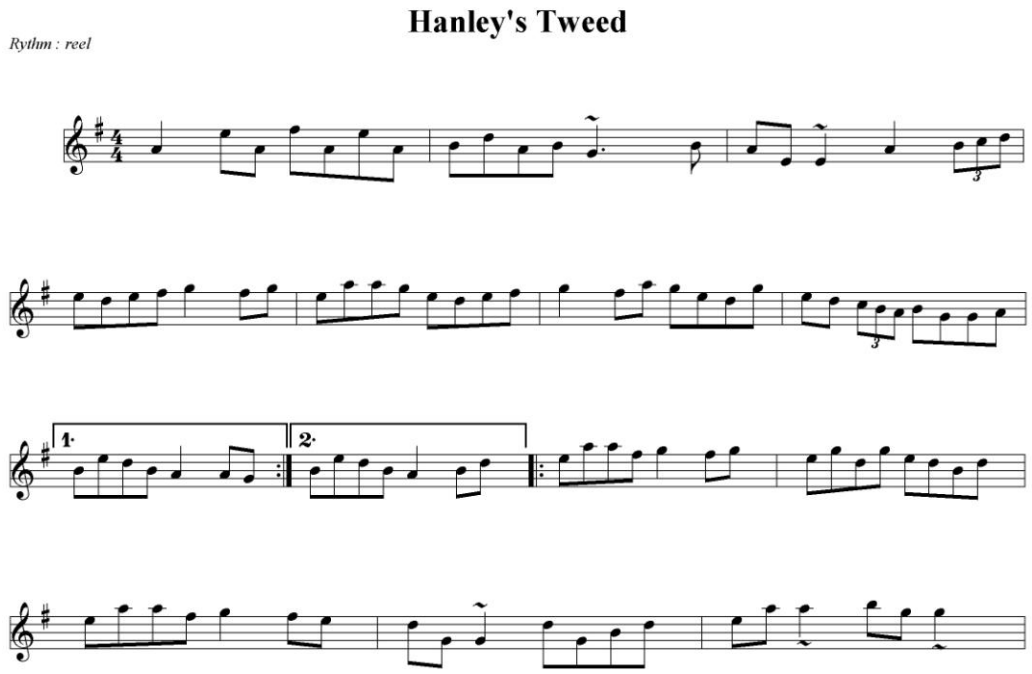


Figure 7: Hanley's Tweed Reel in Staff Notation. Source: Author

Music is also available in various electronic forms such as MP3 (Moving Picture Experts Group 1992), Windows Media Audio (WMA) and MIDI for example. However, none of these are text based.

ABC Notation is a language designed by Chris Walshaw in 1995 to transcribe music in text notation (Walshaw 1995) . Title, musical key, time signature and musical notes are described using ABC Notation and stored in text files with an abc extension.

Table 5: Paddy Keenan's Jig in ABC Notation

```
X: 1
T: Paddy Keenan's
M: 6/8
L: 1/8
R: jig
K: Edor
D|EGA B2A |Bee B2A |GBB FAA |GFE FED
EGA B2A| Bee B2A |GBB FAA |GEDE2D:|E|
Bef gfe| fgf edB |AFF dAF |AB=c dBA|
```

Bef gfe| fgf edB |AFF dAF |FED E2:|

Table 5 shows how Paddy Keenan's jig would be represented in ABC Notation using common fields X, T, M, L, R and K as outlined below;

- X represents the sequence number of the tune in the abc file. ABC notation supports multiple tunes per file and each is numbered sequentially.
- T represents the title of the musical piece. Multiple T fields may be specified within the ABC file representing the different titles a musical piece may have.
- M is the measure or time signature of the piece.
- L is the length of each musical note.
- R is the type of tune e.g. reel, jig, hornpipe.
- K represents the musical key of the tune.

These header fields are followed by the musical notes of the tune.

2.4.1 Why ABC Notation?

Irish traditional music databases in ABC format were chosen for use in this project for the following reasons;

- ABC notation is text based and lends itself to being easily parsed by computer.
- ABC notation can easily be stored in a relational database.
- Thousands of Irish traditional dance music tunes are freely available in the ABC format.
- The ABC specification supports musical key, tune title, time signature and other fields necessary to perform string distance experiments.

2.5 Conclusion

This chapter began by exploring what music comparison is. It outlined how Parsons Code, invented by Denys Parsons in the 1970's, uses the concept of melodic contours to compare musical sequences. It explains how the Melodic Indexing System, designed by Breandán Breathnach in the 1960's, uses a transposition invariant code to assess the similarity of two pieces of music. This chapter also examined the

advantages and disadvantages of both methods. A method of calculating normalised scores for both systems was explained in detail. The accuracy of results rankings was identified as an area for future work and further development. A brief introduction to ABC notation was given along with a short overview of why ABC notation was chosen for this project.

3. STRING DISTANCE ALGORITHMS

3 Introduction

There are numerous string distance algorithms available for a variety of purposes including DNA comparison and spelling checks for instance. These algorithms are normally used to calculate how similar or dissimilar two strings are. This chapter outlines what similarity means in the context of this project, some uses for music similarity and how music theory was considered when evaluating string distance algorithms. This chapter also looks at the work of three world experts, Levenshtein, Winkler and Lemström and how they use three different methods to calculate the distance between two strings of text.

3.1 Choosing a suitable algorithm

Many different string distance algorithms are available and were evaluated briefly before deciding on potential candidates for the purpose of conducting string distance experiments on musical data. These included algorithms such as the Levenshtein algorithm (Levenshtein 1966) which is used to measure edit distance between two strings, the Jaro-Winkler algorithm (W. E Winkler 1999) used in spell checkers to identify misspelled words, the Damerau-Levenshtein algorithm (Damerau 1964), a variation on the original Levenshtein algorithm that supports horizontal transpositions, Hamming distance (Hamming 1950), which measures the amount of substitutions it takes to transform one string into another of equal length and the SIA(M)ESE algorithm (Wiggins et al. 2002), a transposition invariant method of retrieving musical patterns in polyphonic musical databases.

3.1.1 Definition of similarity

In their paper, *Cognitive Adequacy in the Measurement of Melodic Similarity: Algorithmic vs. Human Judgments* (Muellensiefen & Frieler 2003, p.4), Müllensiefen and Frieler define a similarity measure as the mapping of the abstract space of two melodies on a value between 0 and 1. They also state that a similarity measure should be normalised and a melody mapped to itself should have a similarity of 1.

Humans do not always agree what similarity means in the context of music. Allan and Wiggins (Allan & Wiggins 2006) identified that listeners place significance on different features of music they regard as being important for the purposes of similarity.

Holzapfel (Holzapfel & Stylianou 2010) proposes that a morphological approach utilising timbre, rhythmic and melodic characteristics of traditional music be used in the assessment of similarity. The matrix constructed in Section 6.4 uses such a morphological approach by combining four different methods of assessing similarity to return an overall similarity score.

3.1.2 Uses of similarity measures

Measuring similarity in music has numerous applications. Eerola et al. suggest that folk melodies can be classified and categorised by calculating the city block distance between statistical measures taken from each melody (Eerola et al. 2000). Amazon, eBay and other online retailers often use similarity algorithms to identify potential products to offer shoppers via a recommendation system.

Similarity is also used in the context of law suits for the assessment of infringement of copyright or intellectual property theft (Cronin 1998).

3.1.3 Music theory considerations

Strings of musical notes have a different structure than strings containing DNA sequences or phrases of words, for example. Different features of string distance algorithms are more appropriate for evaluating how similar one sequence of musical notes is to another.

While evaluating algorithms, particular attention was paid to those algorithms with features that could be applied to strings of musical notes. For example, the following features were identified in the Levenshtein, Jaro-Winkler and Lemström Semex algorithms;

- When comparing strings of musical notes, the Levenshtein algorithm returns a measure of how many edits it would take to convert one sequence of notes into another using insertions, deletions and substitutions i.e. adding, subtracting or replacing notes until a sequence of notes is converted into the target sequence of notes.
- The horizontal transposition feature of the Jaro-Winkler algorithm allowed for the fact that notes could be played in different sequences (referred to as variations in Irish music).
- It is common for Irish traditional dance tunes to have a few introductory notes before the melody is played. This feature of Irish music is similar to the concept of prefixes as described in the Jaro-Winkler algorithm.
- The Semex algorithm was designed for the purpose of comparing strings of music notes. It allows for transposition invariant searches and also for searching for sub-sequences of notes. For these two reasons it was chosen to compare Irish traditional music tunes in ABC notation format.

3.2 The Levenshtein Algorithm

In 1965, the Russian academic Vladimir Levenshtein proposed a metric for calculating the distance between two strings (Levenshtein 1966). The article was first published in English in 1966. Levenshtein proposed that the distance between two strings of text could be measured by counting the minimum number of edits it would take to change one string into another using only insertions, deletions or substitutions. The following example illustrates how the word “goal” could become the word “post” using substitutions only.

Table 6: Levenshtein substitution example

Edit Distance	**Text strings**			
0	G	O	A	L
1	P	O	A	L
2	P	O	A	T
3	P	O	S	T

According to Levenshtein the edit distance between “goal” and “post” is 3. Table 7 shows how the minimum edit distance between two words of different lengths can be calculated using substitutions and one insertion.

Table 7: Levenshtein insertion example

Edit Distance	**Text strings**				
0	B	A	R	K	
1	B	A	R	K	L
2	G	A	R	K	L
3	G	R	R	K	L
4	G	R	R	W	L
5	G	R	O	W	L

As can be seen from the example in Table 7 the four character word “BARK” has an edit distance of 5 from the five character word “GROWL”. The character “L” was inserted after the character “K” in “BARK” at a cost of 1. Similarly, if calculating the edit distance in reverse, from “GROWL” to “BARK”, removing any character to turn “GROWL” from a five character word into a four character word would also have a cost of 1.

Dynamic programming techniques are frequently used to construct computer algorithms for calculating Levenshtein distance between two strings of text. A two dimensional array is created equal in size to the product of the length of both strings. This array is then used to form a matrix with each location holding edit distance values. The costs of previous calculations are carried over to the next calculation.

Table 8 shows how the edit distance between the text strings “Sunday” and “Monday” are calculated using a matrix. One substitution is required to change “S” to “M”, the cost of which is 1. This cost is carried over to the next comparison. One substitution is required to change “u” to “o”, also at a cost of 1. Therefore, the total cost of changing “Su” to “Mo” is 2. The comparison process continues until all locations have been calculated. The minimum Levenshtein edit distance is the value held in the bottom right cell of the matrix.

Table 8: Calculating Levenshtein edit distance using a matrix

		S	**u**	**n**	**d**	**a**	**y**
	0	1	2	3	4	5	6
M	1	**1**	2	3	4	5	6
o	2	2	**2**	3	4	5	6
n	3	3	3	**2**	3	4	5
d	4	4	4	3	**2**	3	4
a	5	5	5	4	3	**2**	3
y	6	6	6	5	4	3	**2**

While this process provides a mechanism for constructing implementations of the Levenshtein algorithm it is not very efficient for large strings as the number of comparisons and memory requirements increase with the length of the text strings.

Table 9 shows an implementation by Chas Emerick (Emerick 2003) that is more efficient for larger string comparisons by using two single dimension arrays equal to the sum of 2 + the lengths of both strings instead of a much larger two dimensional array.

Table 9: Java implementation of Levenshtein edit distance using dynamic programming techniques (Emerick 2003)

```
public static int getLevenshteinDistance (String s, String t) {
  if (s == null || t == null) {
    throw new IllegalArgumentException("Strings must not be null");
  }

  int n = s.length(); // length of s
  int m = t.length(); // length of t

  if (n == 0) {
    return m;
  } else if (m == 0) {
    return n;
  }

  int p[] = new int[n+1];
  int d[] = new int[n+1];
  int _d[];

  int i;
  int j;
```

```
  char t_j;

  int cost;

  for (i = 0; i<=n; i++) {
     p[i] = i;
  }

  for (j = 1; j<=m; j++) {
     t_j = t.charAt(j-1);
     d[0] = j;

     for (i=1; i<=n; i++) {
        cost = s.charAt(i-1)==t_j ? 0 : 1;
        d[i] = Math.min(Math.min(d[i-1]+1, p[i]+1),  p[i-1]+cost);
     }

     _d = p;
     p = d;
     d = _d;
  }

  return p[n];
}
```

The Levenshtein implementation in Table 9 was used to calculate the Levenshtein edit distances for the experiments in Section 6.2.4.

3.3 The Jaro-Winkler Algorithm

In 1971 Matthew A. Jaro introduced UNIMATCH (UNIversal MATCHer), a system of linking US census records that used the concept of weighting parameters in order to increase the confidence level in a possible census record match. The more unusual the data that is matched the less likely the match is accidental. For example, if two social welfare records match because they have the same surname, "Murphy" this match is more likely to be correct if some other unusual piece of information also matches such as a dependents name or date of birth (M. A Jaro 1971, pp.526-527).

Five years later Jaro introduced a method of comparing strings that utilised insertions, deletions and transpositions (M. A Jaro 1976) and this was further refined in 1989 when the U.S. Bureau of the Census processed records for the 1985 census of Tampa, Florida (MA Jaro 1989).

In 1999, William E. Winkler, also of the U.S. Bureau of the Census claimed that a modified version of the Jaro distance metric showed a considerable improvement over

instances where exact character matching was used (W. E Winkler 1999). Winkler also claims that in a study of twenty string comparison techniques by C.D. Budzinsky, the Jaro distance metric was second best and the improved Jaro-Winkler version was best (Budzinsky 1991). The improved Jaro-Winkler algorithm was used in this project in order to carry out the experiments in Section 6.2.5.

Both Jaro and Jaro-Winkler distances are expressed as values between 0 and 1. A score of 0 means that both strings are completely different and a score of 1 means that both strings are identical. Values between 0 and 1 indicate a measure of how similar strings of text are.

The Jaro distance d_j between strings *s1* and *s2* is calculated using the following formula;

$$d_j = \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t}{m}\right)$$

Figure 8 : Jaro Distance Formula

Where m is the number of matching characters, |s1| is the length of string 1 and |s2| is the length of string 2 and t is the number of transpositions. A transposition is a character match out of sequence within a distance of one less than half the length of the longest string. For example, the strings “there” and “tehre” have two transposition matches. The character “h” in position 2 in the string “there” matches the character “h” in position 3 in the string “tehre”. This transposition match has a distance of 1 and this is less than half the length of the longest string minus 1 (a distance of 1.5 in this case). Similarly, the character “e” in position 3 in the string “there” matches the character “e” in position 2 in the string “tehre”. The maximum distance for a transposition match to be valid may be expressed as;

$$\left\lfloor \frac{\max(|s1|,|s2|)}{2} \right\rfloor - 1$$

Figure 9: Jaro Distance Transposition Formula

William E. Winkler's modification introduces the concept of weighted prefixes so that Jaro-Winkler distance d_w may be expressed as

$$d_w = d_j + (\ell p(1 - d_j))$$

Figure 10: Jaro-Winkler Formula

Where dj is the Jaro distance between two strings s1 and s2, ℓ is the length of an identical prefix in string 1 and string 2 and p is the weight given for having a matching prefix.

Given the text strings of musical notes "BEFGFEFGFEDBAFFDAFFEDEEE" and "BEFGFEFFFEDBBAFDAFFEDEEE" the Jaro-Winkler distance may be calculated as follows;

Table 10: Matches & transpositions between two strings of notes

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
S1	B	E	F	G	F	E	F	G	F	E	D	B	**A**	**F**	F	D	A	F	F	E	D	E	E	E
S2	B	E	F	G	F	E	F	**F**	F	E	D	B	B	**A**	F	D	A	F	F	E	D	E	E	E
	m	m	m	m	m	m	m	**t**	m	m	m	m	**t**		m	m	m	m	m	m	m	m	m	m

The length *L*, of both strings, is 24.

There are 21 matches *m*.

There are 2 transpositions *t.* Character "A" in position 13 in string 1 is a transposition match for character "A" in position 14 in string 2. Character "F" in position 14 in string 1 is a transposition match for character "F" in position 8 in string 2.

There is 1 non-matching character.

Substituting these values into the formula in Figure 8, the Jaro distance is calculated as follows;

$$\frac{1}{3}\left(\frac{21}{24} + \frac{21}{24} + \frac{21-2}{21}\right)$$

Figure 11: Jaro distance calculation

The Jaro distance is therefore 0.885 (correct to three decimal places). In order to calculate the Jaro-Winkler distance we substitute appropriate values into the formula in Figure 10. Figure 11 shows how the Jaro distance is calculated for the strings in Table 10. Winkler suggests a maximum of 4 for the length of the common prefix *l* and a default value of 0.1 (up to a maximum 0.25) for the weight *p* (W. E Winkler 1999).

$$d_w = 0.885 + \left(4 \times 0.1(1 - 0.885)\right)$$

Figure 12: Jaro-Winkler distance calculation

The Jaro-Winkler distance d_w is therefore 0.931 (correct to three decimal places).

The freely available Java implementation of the Jaro-Winkler algorithm by Lingpipe (Carpenter 2010) was used to conduct experiments in Section 6.2.5.

3.4 The Lemström Semex algorithm

In their paper *SEMEX - An Efficient Music Retrieval Prototype*, Kjell Lemström and Sami Perttu introduced fast and efficient bit-parallel algorithms for retrieving music that were transposition invariant (K Lemström & Perttu 2000).

The Lemström Semex (Search Engine for Melodic Excerpts) algorithm accepts two parameters, a pattern to search for and a large string within which the search is performed. Both parameters accept arrays of integers which represent musical notes. The purpose of this algorithm is to find the longest common subsequence between a pair of musical sequences. This subsequence could be an exact match, a transposed match or an approximate match. According to Lemström and Ukkonen (K Lemström & Ukkonen 2000, sec.6), the longer a common subsequence is, the greater the similarity between both sequences.

Table 11: Lemström Semex Java method by Dr. Bryan Duggan

```
public static float minEdSemex(int[] pattern, int[] text)
    {
        int pLength = pattern.length;
```

```
        int tLength = text.length;
        int difference = 0;

        int sc;

        if (pLength == 0)
        {return -1;}
        if (tLength == 0)
        {return -1;}

        int[][] d = new int[pLength + 1][tLength + 1];

        // Initialise the first row and column
        for (int i = 0; i < tLength + 1; i++)
        {d[0][i] = 0;}
        for (int i = 0; i < pLength + 1; i++)
        {d[i][0] = i;}

        for (int i = 1; i <= pLength; i++)
        {
            sc = pattern[i - 1];
            for (int j = 1; j <= tLength; j++)
            {
                int v = d[i - 1][j - 1];
                if (j - 2 < 0 || i - 2 < 0)
                {difference = 1;}
                else if ((text[j - 1] - text[j - 2]) != (pattern[i -
1] - pattern[i - 2]))
                {difference = 1;}
                else
                {difference = 0;}
                d[i][j] = Math.min(Math.min(d[i - 1][j] + 1, d[i][j -
1] + 1), v + difference);
            }
        }
        int[] lastRow = d[pLength];
        int min = Integer.MAX_VALUE;
        for (int i = 1; i < tLength + 1; i++)
        {
            int c = lastRow[i];
            if (c < min)
```

```
            {min = c;}
        }
        return min;

    }
```

Since Lemström and Perttu proposed the Semex prototype in 2000, Lemström has collaborated with other computer scientists doing research on the Longest Common Sequence (LCS) problem, most notably with Navarro and Pinzon in 2004 in an article entitled *Practical algorithms for transposition-invariant string-matching* (K. Lemström et al. 2005). In this article Lemström et al. propose improvements specifically designed to provide performance increases over classical distance algorithms. A branch and bound method of identifying transposition invariant sequences along with a bit-parallel algorithm capable of handling more complex sub-sequences is presented.

3.5 Conclusion

This chapter outlined what a string distance algorithm is and explains what (dis)similarity means in the context of this project. This chapter explains that various algorithms were evaluated with regard to their suitability for calculating the distance between two strings of musical notes. Features that could be applied to strings of musical notes were identified. Finally, the Levenshtein, Jaro-Winkler and the Lemström Semex algorithms were explained in detail.

4. IMPROVED ALGORITHMS & A RANKING SYSTEM

4 Introduction

The purpose of this chapter is to outline three contributions to the body of knowledge in the field of music comparison using string distance algorithms. It shows how two features of the Jaro-Winkler string distance algorithm may be used to weight "out of sequence" musical notes (transpositions) and introductory notes (prefixes). This chapter also outlines the basis for a ranking system that combines the results from multiple algorithms to give a single similarity score and standard deviation.

4.1 Modifications to the Jaro-Winkler Algorithm for Irish music

During the evaluation of string distance algorithms suitable for performing similarity comparisons on ABC notation data of Irish traditional dance tunes it became apparent that the Jaro-Winkler algorithm had two unique characteristics that could have a practical application towards identifying similar Irish music phrases.

4.1.1 Horizontal Transpositions

Unlike the Levenshtein algorithm, the Jaro-Winkler algorithm allows characters out of sequence to be transposed. The algorithm's scoring mechanism weights characters within a distance of half the length of the longest string minus 1. The formula to calculate the correct transposition distance can be seen in Figure 9. Transpositions are frequently used for identifying spelling mistakes as out of sequence characters are weighted higher than incorrect characters so that an incorrectly spelled word will score almost as high as the correctly spelled version of the word. Consider the following example;

Table 12: Jaro-Winkler Transposition Example

1	2	3	4	5	6	7
I	R	**L**	**E**	A	N	D
I	R	**E**	**L**	A	N	D

Table 12 shows how the letter "E" in position 3 of the word "IRELAND" is mapped to "E" in position 4 of the word "IRLEAND". Similarly, "L" in the correctly spelled word is mapped to the L in the incorrectly spelled word. A comparison between the Levenshtein and Jaro-Winkler algorithms shows that the Levenshtein distance is 2 edits and when normalised for the length of the strings this represents a score of 0.714 (0 being completely different and 1 being a perfect match). The Jaro-Winkler algorithm scores this pair of strings as being 0.962 (0 being completely different and 1 being a perfect match) which is higher than Levenshtein as it allows scores for horizontal transpositions. The Levenshtein algorithm classifies the characters "E" and "L" as completely incorrect giving both a cost of 1 each, whereas, the Jaro-Winkler algorithm lessens this cost because the characters are correct but out of sequence and scoring them almost as high as correct "in sequence" characters.

Speaking about Irish music in his article *Style in Traditional Irish Music* (McCullough 1977, p.85), Lawrence McCullough states that individual pieces of Irish music have been completely reshaped by musicians. He indicates that there are four main factors involved;

- Ornamentation, a process of embellishing individual notes
- Variation in melodic and rhythmic patterns
- Phrasing – choosing where to include rests or short pauses
- Articulation, how notes are played together. Examples of articulation are
    - Slur – when a note slides into the next note without separation.
    - Staccato – when notes are separated by short rests in between each note.
    - Legato – when notes are played smoothly together.

The following contribution specifically relates to McCullough's second assertion, variation in melodic patterns. The Jaro-Winkler algorithm could be used to weight an out of sequence series of musical notes so that it scores almost as highly as the correct sequences of notes.

4.1.2 **Contribution 1:** Weighting melodic sequence variation

The transposition feature of the Jaro-Winkler algorithm can be adapted to recognise certain melodic variations that McCullough writes about. Specifically, the algorithm was adapted to give weight to out of sequence notes within a distance calculated with respect to the time signature of the piece of music. Consider the following example, a jig called *"Wallop the Spot"* available on an audio recording of the group Osna (Osna 1999, Track.12). The opening phrase of the jig is normally played as follows;

Table 13: Standard opening phrase of the Wallop the Spot jig

FEF DFA BAF DDD

On track 12, the whistle player swaps notes 1 & 2 and notes 7 & 8, reshaping the standard phrase so that it becomes;

Table 14: Reshaped opening phrase of the Wallop the Spot jig (Osna 1999)

EEF DFA ABF DDD

The Jaro-Winkler algorithm was altered so that the proximity method accepted an extra parameter – searchRange. Specific values related to the time signature of the comparison strings were passed to this parameter, for example, 3 was passed for jigs and 4 for reels.

Table 15: Adapted Jaro-Winkler method with searchRange parameter

```
public double proximity(CharSequence cSeq1, CharSequence cSeq2, int
searchRange) {
        int len1 = cSeq1.length();
        int len2 = cSeq2.length();
        if (len1 == 0)
            return len2 == 0 ? 1.0 : 0.0;
        boolean[] matched1 = new boolean[len1];
        Arrays.fill(matched1,false);
        boolean[] matched2 = new boolean[len2];
        Arrays.fill(matched2,false);

        int numCommon = 0;
```

```
        for (int i = 0; i < len1; ++i) {
            int start = Math.max(0,i-searchRange);
            int end = Math.min(i+searchRange+1,len2);
            for (int j = start; j < end; ++j) {
                if (matched2[j]) continue;
                if (cSeq1.charAt(i) != cSeq2.charAt(j))
                    continue;
                matched1[i] = true;
                matched2[j] = true;
                ++numCommon;
                break;
            }
        }
        if (numCommon == 0) return 0.0;

        int numHalfTransposed = 0;
        int j = 0;
        for (int i = 0; i < len1; ++i) {
            if (!matched1[i]) continue;
            while (!matched2[j]) ++j;
            if (cSeq1.charAt(i) != cSeq2.charAt(j))
                ++numHalfTransposed;
            ++j;
        }
        int numTransposed = numHalfTransposed/2;

        double numCommonD = numCommon;
        double weight = (numCommonD/len1
                         + numCommonD/len2
                         + (numCommon -
numTransposed)/numCommonD)/3.0;

        if (weight <= mWeightThreshold) return weight;
        int max =
Math.min(mNumChars,Math.min(cSeq1.length(),cSeq2.length()));
        int pos = 0;
        while (pos < max && cSeq1.charAt(pos) == cSeq2.charAt(pos))
            ++pos;
        if (pos == 0) return weight;
        return weight + 0.1 * pos * (1.0 - weight);
    }
```

Comparing these two strings with the Levenshtein algorithm gives an edit distance of 3 and a normalised score of 0.75. The Jaro-Winkler score weights the out of sequence notes and gives a score of 0.85. The transposed out of sequence characters can be seen in Table 16.

Table 16: Jaro-Winkler transpositions for a Wallop the Spot variation

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
F	E	F	D	F	A	B	A	F	D	D	D
E	E	F	D	F	A	A	B	F	D	D	D

The "B" in column 7 of the standard phrase (row 1) is transposed horizontally with the "B" in column 8 of the reshaped phrase. Similarly, the "A" in column 8 of the standard phrase is transposed with the "A" in column 7 of the reshaped phrase. Note that the "F" in column 1 of the standard phrase cannot be transposed with the "F" in column 3 of the reshaped phrase as column 3 contains correct notes that are already matched with each other.

It is worth noting that a flaw in Breandán Breathnach Melodic Indexing System is exposed by the example in Table 16. The indexing code for the standard phrase is 33156311 and the reshaped phrase 23155311. Because the first note of each phrase is different the first digit of the eight digit indexing codes are also different. This means that when the index cards are stored numerically they will not be in proximity. In this case, the Jaro-Winkler algorithm correctly identifies that both phrases are similar, scoring higher than both the Levenshtein and Breathnach methods.

4.1.3 **Contribution 2:** Weighting tune prefixes

The Jaro-Winkler algorithm supports weighted prefixes of up to four characters long. On occasion, Irish traditional dance tunes have a two note prefix that is played as an introduction to the tune. This prefix is omitted when the tune is repeated.

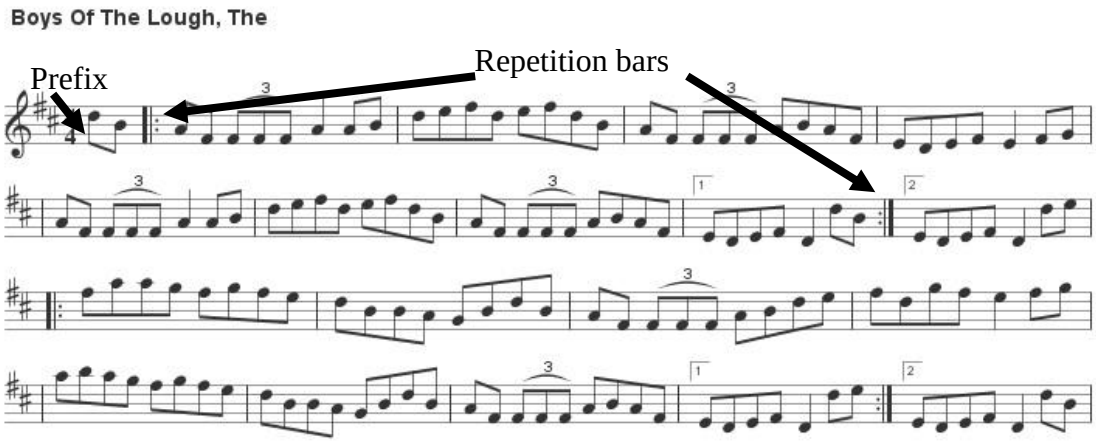


Figure 13: Boys of the Lough with prefix and repetition bars. Source: Author

Figure 13 shows a two note prefix for the reel "The Boys of the Lough". Not all Irish dance tunes have prefixes but if one exists, it will always precede the opening repetition bar. Fortunately, ABC notation supports the inclusion of prefixes in its specification as can be seen in Table 17.

Table 17: Boys of the Lough with prefix in ABC Notation. Source (Lonelyhearts 1978)

```
X: 1
T: Boys Of The Lough, The
M: 4/4
L: 1/8
R: reel
K: Dmaj
dB|:AF (3FFF A2 AB|defd efdB|AF (3FFF ABAF|EDEF E2 FG|
|AF (3FFF A2 AB|defd efdB|AF (3FFF ABAF|1 EDEF D2 dB:|2 EDEF D2 de|
|:faag fgfe|dBBA GBdB|AF (3FFF ABde|fdgf e2 fg|
|abag fgfe|dBBA GBdB|AF (3FFF ABAF|1 EDEF D2 de:|2 EDEF D2 dB|
```

Two note prefix

The two note prefix is determined by factors such as the musical key and first note of the tune. Usually the two notes that comprise the prefix will be in close proximity to the first note of the tune and will either descend or ascend towards it. Table 18 gives some examples of prefixes to Irish dance tunes.

Table 18: Example prefixes for Irish tunes

Prefix	Tune Body
gg|:	fdAF GECE
gf |:	edBA GEDE
ag |:	EAAA BGAG

If the prefixes are the same does this represent a greater likelihood that the tunes are similar? In his articles, *The State of Record Linkage and Current Research Problems* (W. E Winkler 1999, p.7) and *Overview of Record Linkage and Current Research Directions* (WE Winkler 2006, p.35), William E. Winkler states that two records that agree on a rarely occurring feature are more likely to represent a match than frequently occurring features. Similarly, Breandán Breathnach's Melodic Indexing System derived indexing codes exclusively from the first sixteen notes of each tune. Both methodologies clearly place importance on the beginnings of strings whether they consist of text or musical notes.

For the purposes of the Jaro-Winkler experiments the default values of a 4 character prefix with a 0.25 weighting were observed. In order to fully test if these values were suitable the corpus of ABC notated Irish dance tunes would have to be examined in depth to ensure the following;

- That prefixes were entered according to the ABC notation specification.
- The minimum and maximum length of prefixes.
- Any discernable rules regarding prefixes in Irish music that could be further incorporated into the Jaro-Winkler algorithm.
- The most appropriate lengths for prefixes of musical notes.
- The correct weight to afford to prefixes

Winkler suggests that weighting should not exceed 1. For example, a 4 character prefix with a weighting of 0.25 results in a maximum weighting of 1 as 4 x 0.25 = 1. A two note prefix would have a maximum weight of 0.5 as 2 x 0.5 = 1.

The testing of Jaro-Winkler prefixes is reserved for future work and research in Section 7.4.

4.2 Improvements to the Levenshtein algorithm

One of the drawbacks of the Levenshtein algorithm is that it is not capable of key invariant or time signature invariant comparisons. In spite of this, it remains one of the most popular string distance algorithms for musical comparison.

Rather than integrate a vertical transposition invariant feature into the Levenshtein algorithm it is possible to convert both sequences of notes to relative or absolute intervals and compare these sets of intervals using an unaltered Levenshtein algorithm.

It is also possible to reduce musical pieces to a common time signature before performing a Levenshtein comparison. All of the tune parts imported into the corpus for the purpose of performing string distance experiments had a 2/4 version derived from the original sequence of notes and this was stored along with relative and absolute intervals calculated using the musical key.

Table 19: Example of how tune parts are stored in the corpus database

Field	Value
Name	Longacre, The
Notes	AAEDDECCEDDECCEE
2/4 Version	ADDCEECE
Semex Intervals	0,-3,-1,0,1,-2,0,2,-1,0,1,-2,0,2,0
MIC Intervals	1443553

A horizontal transposition feature as in the Jaro-Winkler algorithm could help improve the accuracy of the Levenshtein algorithm. A variant of the Levenshtein algorithm already exists called the Damerau-Levenshtein algorithm that allows for transpositions. It is a hybrid of the system proposed by Frederick J. Damerau for spelling mistakes requiring one edit operation (Damerau 1964) and the Levenshtein algorithm.

The conclusions drawn from examining any possible improvements to the Levenshtein algorithm were as follows;

- Required features such as horizontal and vertical transpositions were already available in the Jaro-Winkler and Lemström algorithms.
- A time signature invariant feature would be better performed outside of the algorithm. For example, data would be pre-processed before performing a Levenshtein comparison between two strings of notes.
- The unaltered Levenshtein edit distance algorithm has value and remains a popular method for music comparison.

4.3 Prototype for a Combined Ranking System

Having identified the strengths and weaknesses of each of the string distance algorithms and after implementing a framework for generating metrics regarding tune parts held in a corpus the author felt that combining multiple methods and algorithms could be used to define a combined similarity scoring system.

4.3.1 **Contribution 3:** Combined Ranking Scores

In order to combine various algorithms a ranking system was first developed. This involved running four separate SQL queries and ordering the results in descending order by algorithm. The Levenshtein and Jaro-Winkler algorithms were run on the unaltered notes of the tune parts and sequences with non-dominant notes removed (referred to as 2/4, 24 or TWOFOUR data in this project). The results were stored in a relational database.

Figure 14 shows the first twenty rows of tune comparisons between tune id 8425 and various others along with the algorithm scores abbreviated as LEVEN, JARO, LEVEN24 and JARO24. These rows are sorted by Levenshtein score in descending order.

#	ID	TUNE_A_ID	TUNE_B_ID	LEVEN	JARO	LEVEN24	JARO24
1	84481	8425	17825	0.5625	0.8094135802469135	0.5625	0.8094135802469135
2	84463	8425	14383	0.5	0.7083333333333334	0.5	0.7083333333333334
3	84451	8425	11232	0.46875	0.7836538461538461	0.46875	0.7836538461538461
4	84448	8425	9948	0.46875	0.8138888888888888	0.46875	0.8138888888888888
5	84486	8425	18069	0.4375	0.7724358974358975	0.4375	0.7724358974358975
6	84485	8425	18068	0.4375	0.798076923076923	0.4375	0.798076923076923
7	84475	8425	17461	0.4375	0.760576923076923	0.4375	0.760576923076923
8	84446	8425	9854	0.4375	0.7836538461538461	0.4375	0.7836538461538461
9	84488	8425	18763	0.40625	0.7723765432098766	0.40625	0.7723765432098766
10	84480	8425	17824	0.40625	0.8392857142857144	0.40625	0.8392857142857144
11	84476	8425	17569	0.40625	0.7007575757575758	0.40625	0.7007575757575758
12	84490	8425	19178	0.375	0.7208333333333333	0.375	0.7208333333333333
13	84483	8425	17972	0.375	0.606359649122807	0.375	0.606359649122807
14	84469	8425	15843	0.375	0.768840579710145	0.375	0.768840579710145
15	84466	8425	14654	0.375	0.7625000000000001	0.375	0.7625000000000001
16	84464	8425	14493	0.375	0.78475	0.375	0.78475
17	84455	8425	12400	0.375	0.6856060606060606	0.375	0.6856060606060606
18	84452	8425	11547	0.375	0.7208333333333333	0.375	0.7208333333333333
19	84450	8425	11231	0.375	0.75	0.375	0.75
20	84449	8425	9950	0.375	0.6965579710144928	0.375	0.6965579710144928

Figure 14: Levenshtein ordered ranking system for tune comparisons. Source: Author

The columns in Figure 14 from left to right represent the following;

- # represents rank.
- ID is a unique ID and primary key for the row data.
- TUNE_A_ID represents the first of the pair of tunes compared.
- TUNE_B_ID represents the second of the pair of tunes compared.
- LEVEN represents the Levenshtein edit distance between the unaltered melodies represented by TUNE_A_ID and TUNE_B_ID.
- JARO represents the Jaro-Winkler distance between the unaltered melodies represented by TUNE_A_ID and TUNE_B_ID.
- LEVEN24 represents the Levenshtein edit distance between the 2/4 version of the melodies represented by TUNE_A_ID and TUNE_B_ID.
- JARO24 represents the Jaro-Winkler distance between the 2/4 version of the melodies represented by TUNE_A_ID and TUNE_B_ID.

The rank of a pair combination can be ascertained by using Figure 14. For example, the tune pair with TUNE_A_ID of 8425 and TUNE_B_ID of 17825 has a Levenshtein rank of 1 (row 1 of the database table), tunes 8425 and 14383 have a rank of 2 (row 2 of the database table), tunes 8425 and 17569 would have a Levenshtein rank of 11

(row 11 of the database table). The algorithms were run on a subset of the corpus and the results were stored in a relational database. A prototype Java application was designed in order to compare rankings from each of the algorithms as can be seen in the following diagram.

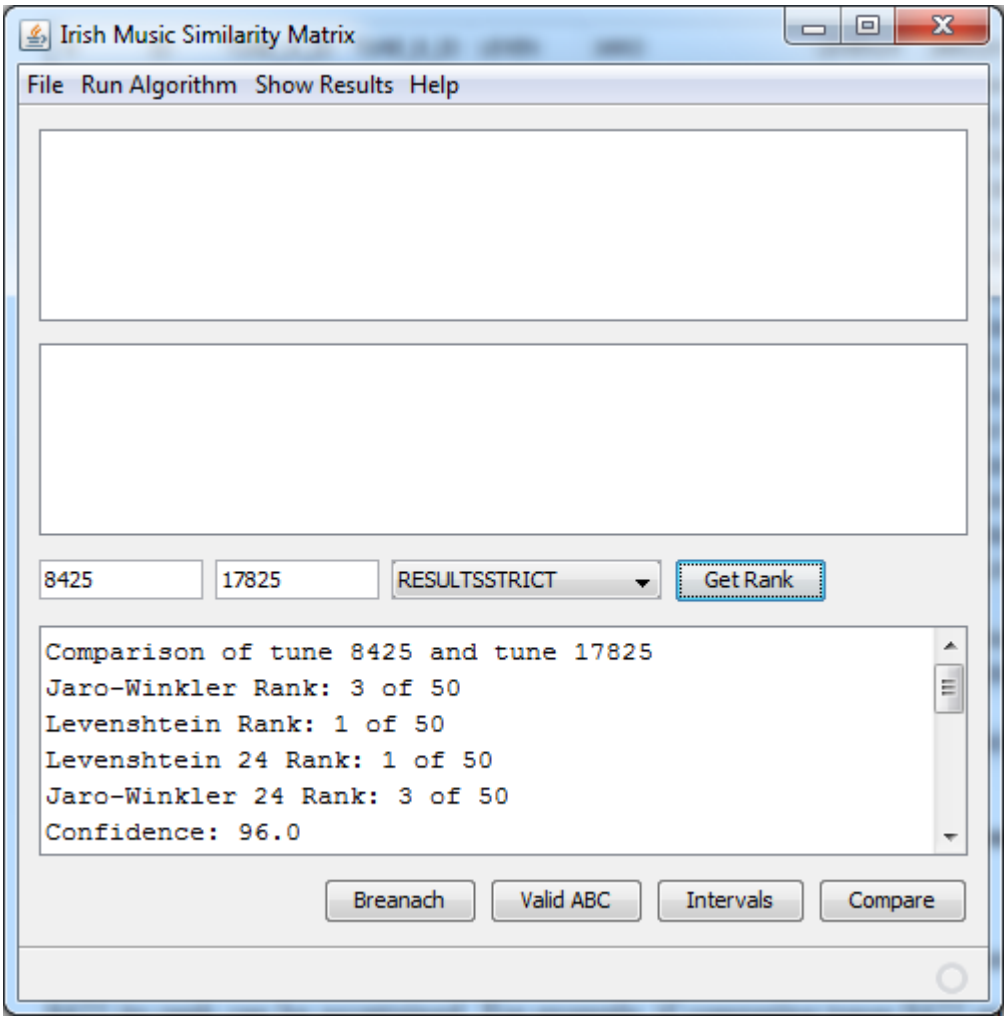


Figure 15: Application used to generate rankings by algorithm

In this example the Levenshtein score is relatively low at 0.5625 but this score is ranked 1st (see Figure 14). The Jaro-Winkler score of 0.8094 is high in comparison to the Levenshtein score and it is also ranked high as the 3rd highest Jaro-Winkler result. Because high rankings were returned by all four comparisons can it be said that two tunes are similar with any degree of confidence? This question is explored in Section 6.4.3.

A visual check of both tunes in the database shows that they are in the same time signature and that both are in the key of D major. Also, 15 notes out of 32 are direct matches and there are a number of candidate notes that could be horizontally transposed.

#	ID	NAME	PART	TUNEKEY	MEASURE	NOTES
1	8425	Con Thadhgo"s	2	Dmaj	2/4	FFEDCAAEFDEDEAAGFDEDCAABCBCEEDDD
2	17825	Braddock"s March	2	Dmaj	2/4	GFEDCBAAFDFDCDEEGFEDCBAAFFACDDDD

Figure 16: Visual comparison of tunes 8425 and 17825

Initially the confidence score was calculated by averaging the ranks and subtracting this from 100%, giving 96% in the above example. Following careful analysis and experimentation in Section 6.2 this result was improved to include the standard deviation between algorithms. This resulted in the delivery of one of the project objectives, defining a process by which a similarity matrix could be constructed. Section 6.4 explains how the similarity matrix was created.

Hypothesis: If multiple different algorithms rank a comparison similarly, can that comparison be assumed to be more accurate than when the algorithms disagree?

In order to test the validity of this hypothesis, a survey of humans was carried out and is described in 6.4.3.

4.4 Conclusion

This chapter outlined three contributions to the body of knowledge - the weighting of melodic variations, the weighting of short prefixes that sometimes prefix Irish traditional music and a method of ranking the results of four algorithms and combining these ranks in order to assess similarity. An assumption was reached that if four different algorithms agreed about the result of a comparison then that comparison could be understood to be more accurate than comparisons where the algorithms disagreed.

5. COMPUTERISING MIC SYSTEM & PARSONS CODE

5 Introduction

In Section 2.2 the work done by Breandán Breathnach was outlined. Section 2.3 explained how Parsons Code could be used to compare musical sequences of notes using melodic contours. In this chapter, flaws in both methods are identified and solutions offered. This chapter outlines contributions 4 and 5 to the body of knowledge.

5.1 Advantages of the Breathnach Melodic Indexing System

As seen in Section 2.2, an incipit from the start of a dance tune is converted into an eight digit Melodic Index Code (MIC) by calculating the intervals between notes in sequence. This code is written on index cards that are stored numerically.

Table 20: Examples of tunes with Melodic Index Codes

Name	Measure	Key	Notes in 2/4	Melodic Index Code
Shlide Aside	12/8	D Major	AFEDEFDD	56611501
Muineira De Casu	6/8	G Major	GEDCDEDB	56611655
Dick The Welshman	2/4	D Major	AFEDFGBA	56621265

Index cards in close physical proximity and numerical order are similar musically according to Breathnach's theory.

5.1.1 Time signature invariant

One of the disadvantages of using string distance algorithms to compare music is that they do not account for musical pieces in different time signatures. Using Breathnach's system, non dominant notes are removed from tune parts, effectively reducing each tune's incipit to a time signature of 2/4. This allows tunes in different time signatures to be compared on an equal basis.

5.1.2 Key invariant

Because the Melodic Index Code is calculated using intervals the Melodic Index Code is key invariant allowing for the comparison of tunes in different musical keys. Similar tunes in different keys can be seen in Table 20.

5.1.3 Easily managed system

Because index cards are stored numerically, fewer comparisons need to be made in order to construct a similarity matrix. As noted in Section 1.4 using string distance algorithms requires the comparison of each tune part with all others in the corpus. Therefore, a corpus of 12,500 Irish traditional tunes having at least two parts each would result in 624,975,000 comparisons (25,000 x 24999). Because Breathnach's system does not require each index card to be compared with all other cards in the system the amount of comparisons that need be made are considerably fewer. One calculation per record is required for Breathnach's system compared to n(n -1) when using string distance algorithms to construct a similarity matrix. This results in a fraction of the computational resources being needed in order to complete a Breathnach similarity matrix compared to a matrix constructed using a string distance method.

During an experiment run as part of the research work described in Section 6.2.6 approximately 49,000,000 comparisons were performed over the course of five days and the results stored in a relational database that reached 2.5 gigabytes (2560 megabytes) in size. By comparison the Breathnach system was completed in minutes and was 40 times smaller, reaching a size of only 64 megabytes.

5.2 Disadvantages of the Breathnach Melodic Indexing System

The Melodic Indexing System performed its function very well in the 1960's and 1970's, identifying duplicates and tunes published in other music collections. As a result, the Ceol Rince na hÉireann tune collections I, II, III, IV & V (Breandán Breathnach 1963) are highly valued by Irish musicians worldwide and are equally as popular as the O'Neill 1001 collection (F. O. &. J. O'Neill 1995). Using the system exactly as designed by Breathnach presents challenges that must be overcome when constructing a music similarity matrix.

5.2.1 Melodic Sequence Variation Anomalies

As seen in Section 2.2 Melodic Index Codes are calculated by discarding non-dominant notes and calculating absolute intervals with reference to a fundamental note. In Section 4.1.2 a disadvantage was identified where an Irish musician reshaped the opening phrase of a tune by playing the notes EEF DFA ABF DDD instead of FEF DFA BAF DDD. These phrases translate to MIC index codes 23155311 and 33156311 respectively. In the corpus of 11,944 tune parts used for this project these versions of the same tune would be stored 1,387 rows apart. In other words, the index cards would not be physically proximate and the duplicate version would not be detected.

5.2.2 Limited Comparisons can be made

Breandán Breathnach's indexing system compared only the very beginnings of each tune. Because ABC data is available for the complete tune, the beginnings of each part of each tune can be compared and indexed. Indeed, the sequence of notes in the whole tune could be converted to a Melodic Index Code and compared. Tunes of the same type were stored with each other. This did not facilitate the easy comparison between jigs, reels, hornpipes, slip jigs and other types of tune.

Figure 17: Storage of Melodic Indexing System. Source: Author

Figure 17 shows how the Melodic Indexing System was stored in the Irish Traditional Music Archive. From top left – Jigs, Reels, Slip jigs/Hornpipes. From bottom left – Jigs2, Reels 2, Polkas/Set Dances/Miscellaneous

5.3 Proposed improvements

Although Brendán Breathnach probably had little computing resources at his disposal in the 1960's when editing his first publication, *Ceol Rince na hÉireann Cuid I* (Breandán Breathnach 1963) his system of Melodic Index Cards lends itself to being converted into a computer algorithm.

5.3.1 **Contribution 4:** Computerisation of the Melodic Indexing System

The implementation of a computerised version of Breandán Breathnach's Melodic Indexing System was constructed in the following manner;

- Irish traditional dance music parts were imported and stored in a relational database. Invalid ABC notation was discarded.
- Parts in ABC notation were converted from various time signatures to a common time signature of 2/4 by programming the Java algorithm in Table 21. The results were stored in the relational database.
- A second Java algorithm (see Table 22) was programmed in order to calculate intervals based on Breathnach's concept of a "fundamental note". Because the tune key is available in each of the ABC tune transcriptions it was used to calculate the fundamental note. Absolute intervals were stored in the same relational database as the corpus of ABC data.

Table 21: Java algorithm to reduce ABC notation to 2/4 time signature. Source: Author

```
public String reduceToTwoFour(String abc, String measure) {
        String two4 = "";
        int counter = 0;
        if  (measure.startsWith("6")  ||  measure.startsWith("9")  ||
measure.startsWith("12")) {
            counter = 3;
        }
```

```
        if (measure.startsWith("4")) {
            counter = 4;
        }
        if (measure.startsWith("2")) {
            return abc; // already in 2/4 time signature
        }

        for (int i = 0; i < abc.length(); i += counter) {
            try {
                two4 += abc.substring(i, i + 1);
                two4 += abc.substring(i + counter - 1, i + counter);
            } catch (Exception e) {
                System.out.println(e.toString());
            }
        }
        return two4;
    }
```

Table 22: Java method for calculating Melodic Index Intervals. Source: Author

```
      public String calculate_BB_Intervals(String input, String key)
{
        key = key.substring(0, 1).toUpperCase();
        input = (input).toUpperCase();
        String control = "CDEFGAB";
        String temp = "";
        int char1, interval, fundamental;
        fundamental = control.indexOf(key);
        for (int i = 0; i < input.length() - 1; i++) {
            try {
                char1 = control.indexOf(input.charAt(i));
                interval = (char1 - fundamental + 1);
                if (interval < 1) {
                    interval += 7; }
                temp += "" + interval;
            } catch (Exception e) {
                System.out.println(e.toString());
            }
        }
        return temp;
    }
```

5.3.2 **Contribution 5:** Compare MIC index codes alphabetically

Breathnach stored the melodic index cards in numerical order using the eight digit code to sort them appropriately. This had the effect of limiting the comparisons that could be done to sequences of notes that were at least sixteen notes long. Sequences of less than 16 notes would result in a melodic index code of less than eight digits meaning that they would not appear in the correct order if sorted numerically.

A simple solution would be to right pad index codes with sufficient 1's to make them eight digits long as in Table 23 below.

Table 23: Index codes with right padded 1's

Index Code
12111111
12121353
14524117
64571156
75441111
77447277

A better solution is to calculate Melodic Index Codes for the whole length of each tune part and storing the results in a database. Sorting the rows alphabetically instead of numerically allows the comparison of incipits of different lengths.

The SQL query in Table 24 sorts rows of tune parts alphabetically, regardless of length as seen in Figure 18.

Table 24: SQL Query to sort tune parts alphabetically

```
select NAME, NOTES, MEASURE, TUNEKEY, BB_INTERVALS from APP.ABC
where BB_INTERVALS is not null order by BB_INTERVALS asc
```

#	NAME	NOTES	MEASURE	TUNEKEY	BB_INTERVALS
1	Tantan"s	EBFBGBEF...	6/8	Emin	125513271255554212661327334452 1
2	Millbrae	AAGBEGBE...	4/4	Amaj	125571376543272212557137654253111255 71...
3	Tullochgorum	CBGDGCFA...	4/4	Cmaj	125612525256125
4	Copenhagen, The	DBEECDDD...	4/4	Dmaj	127152331271554111715234262214 51
5	Jolly Tinker, The	AAABGGGB...	4/4	Ador	127216721172424212721672117242 4
6	Wild Swans at Coole	DECEDCDA...	2/2	Dmix	127217157774234215553555321677765332 32...
7	Long Note, The	DDDECADE...	4/4	Dmix	127222725177154211151125423115 41
8	Butcher"s March (D)	GFAFGEAE...	6/8	Gmaj	1276253211216651
9	False Proof, The	BDCACAGF...	6/8	Bmin	127764554464463712776454444327 1
10	Peeler"s Pocket	GGDBGBDG...	4/4	Gmaj	131111671317643111765322555465 11
11	Paidin O"Rafferty	GBBGGGGB...	6/8	Gmaj	131113421312111513121342353322 16
12	Da Birlie	DDDFDDDD...	4/4	Dmaj	131137777713112347611111727723476126
13	Wild One, The	DEFDGDAG...	6/8	Dmin	131157521311332213115755157452 1
14	Father Kelly"s	DEFFDFED...	4/4	Dmaj	131161123423222343116112353532 7
15	Connie The Soldier	DEFDDEDE...	6/8	Dmaj	131213111321275552312231541452 21
16	Strike the Gay Harp	DEFDFEDA...	6/8	Dmaj	131213151312132513615532111352 2
17	Lord Mounteagle"s	DDAFDGFE...	4/4	Dmaj	131213431316523113121343131652
18	Stay A Wee Bit Bonny Lad	DFAFDFEE...	4/4	Dmaj	131215614122116
19	Old Dudeen, The	BCDDBCDC...	4/4	Bmin	131231154112317431123117551175 57
20	Donald, Willie And His Dog	AACABBCC...	9/8	Amix	13123213132213123231171

Figure 18: Tune parts sorted alphabetically by Melodic Index Code

5.4 Advantages of computerising the Melodic Indexing System

Computerisation of the Breathnach Melodic Indexing System would result in a far superior similarity comparison system for the following reasons;

5.4.1 Larger database of tunes available

Websites like The Session (Keith 2010) allow for members to submit transcriptions of traditional Irish tunes and also many other forms of music. The addition of genre, country of origin or geo-location data could allow for the comparison of tunes across genres or between each country's traditional folk music. For example, relationships or similarities between Irish, English, Scottish, Breton, Galician and Asturian folk music could be identified and explored.

5.4.2 Greater Accuracy

Because computerisation allows for Melodic Index Codes greater than eight digits long as in the original system, the accuracy of the similarity matrix can be increased considerably. Absolute intervals for whole tune parts were calculated and compared instead of comparing 8 digit codes derived from 16 note incipits.

5.4.3 Integration in a Combined Ranking System

In Section 4.3 a confidence scoring system based on the ranking of the results of four string distance algorithms was proposed as Contribution 3. As part of the experimentation and research carried out in Section 6 an algorithm was developed for the calculation of metrics related to the Melodic Indexing System. These metrics included;

- The calculation of the number of rows that separate a pair of tune parts along with the total number of tune parts in the corpus.
- The proximity expressed in the same format as suggested by Muellensiefen & Frieler (Muellensiefen & Frieler 2003) i.e. 0 being perfectly different and 1 being an exact match.

Figure 19 shows how Melodic Indexing System metrics were calculated for the tune parts with id's 8425 and 17825.

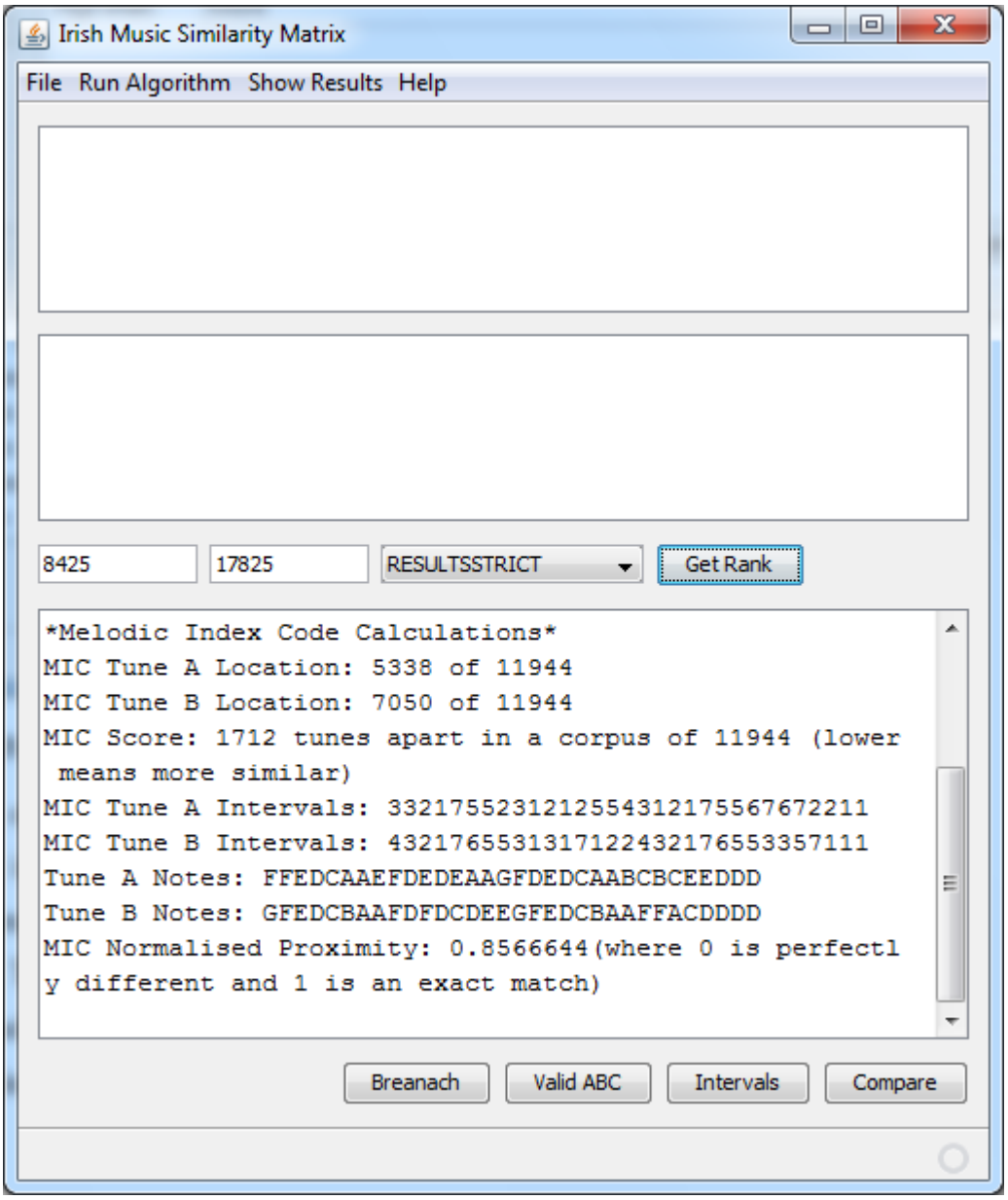


Figure 19: Calculation of Melodic Indexing Metrics

Although the normalised score could be considered high at 0.857 (1 being an exact match and 0 being completely different) it represents a distance of 1712 tune parts in a corpus of 11944. In other words, there are 1711 tune parts more similar to tune part 8425 ascending the matrix to tune part 17825 and possibly others descending from 8425 as can be seen in the following table.

Table 25: Portion of the Melodic Index Code Matrix

ID	**Melodic Index Code**
-----	(MIC codes similar to 8425 removed)
8424	33216221542.....
8425	**33217552312.....**
8426	33222421421.....
-----	(MIC codes similar to 8425 removed)
17824	43216123412.....
17825	**43217655313.....**
17826	44213213211.....

The inclusion of the Melodic Index Code metrics into the confidence / ranking scoring system was completed as part of an experiment in Section 6.4.3.

5.5 Conclusion

The advantages, disadvantages and tradeoffs of Breandán Breathnach's Melodic Indexing System were presented in this chapter. A proposal for the computerisation of the system was presented as contribution 4. Contribution 5 suggests improvements to the system. The use of an alphabetical index rather than a numeric one was suggested in order to overcome the problem of different length melodic index codes.

6. EXPERIMENTATION AND EVALUATION

6 Introduction

The purpose of this chapter is to describe the string distance experiments that were carried out on ABC notation data. Once clean ABC data had been extracted from ABC files contained within music collections referred to in Section 1.7.4 it was stored in a relational database. Java versions of string distance algorithms were obtained and integrated into a programming framework that had been built in order to facilitate the running of experiments. This chapter also describes how two online surveys were carried out and how the hypothesis formed in Section 4.3 was tested. The chapter concludes with a description of how similarity matrices of Irish traditional dance music were constructed.

6.1 Design of experiments

Careful planning went into the design of each experiment. The purpose of carrying out experiments on string distance algorithms was to be able to draw conclusions from analysis of the results. Great care was taken to prevent bias of any kind in the experiments and in the online surveys.

A series of goals in line with the research objectives of this dissertation were formulated and a strategy was formed in order to achieve these goals. The goals were as follows;

- To identify string distance algorithms suitable for Irish traditional music comparison.
- To identify possible areas where string distance algorithms could be improved with respect to music theory.
- To test if humans agreed with the results of string distance algorithm comparisons of Irish music.
- To identify and define a process whereby a Music Similarity Matrix could be constructed for Irish Traditional Music (ITM).

6.2 Experimentation

The following experiments were carried out on clean data held in a relational database.

- Levenshtein edit distance comparisons
- Jaro-Winkler edit distance comparisons
- Lemström Semex distance comparisons
- Melodic Index Code similarity and distance
- Parsons Code similarity and distance
- Ranking and combined scoring
- Various similarity matrix construction experiments

All string distance experiments were carried out on a Dell Inspiron 9400 laptop with an Intel Dual Core 2.0 Ghz processor, 100GB 7200rpm hard drive and 2GB of ram running on the Windows 7 operating system.

6.2.1 Description of raw data

The relational database held a corpus of 11944 tune parts. As the data was imported from ABC text files it was cleaned and pre-processed so that only musical notes remained. This data was obtained from publicly available electronic tune collections mentioned in Section 1.2.5. The Irish music dance tunes were transcribed by users of varying musical ability with over half of the ABC files not validating against the ABC notation specification. Any unreliable data that did not fully comply with the ABC notation specification was immediately discarded.

6.2.2 Pre-processing ABC data

ABC data pre-processing involved the removal of ABC file headers, extra notation, triplet marks, rests, removal of white space and other unnecessary elements. This was achieved using Java methods available in Dr. Bryan Duggan's MattABCTools java class. The cleaned musical note data was stored in the NOTES column of the database. The musical note data was then converted from various time signatures to a 2/4 version and stored in the TWOFOUR column. The relative and absolute intervals between each musical note were also calculated and stored in the INTERVALS and BB_INTERVALS columns. The time signature, musical key and tune part number

were stored in the MEASURE, TUNEKEY and PART columns respectively. All columns were required and any incomplete rows were discarded.

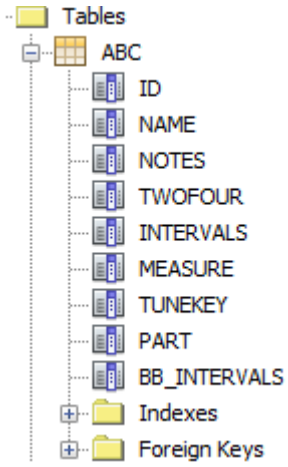


Figure 20: ABC Corpus Schema. Source: Author

- ID – Primary key unique to each row
- NAME – the name of the parent tune
- NOTES - The cleaned notes of the parent tune
- TWOFOUR – First and last notes of each beat preserved
- INTERVALS – NOTES represented as relative intervals
- MEASURE – The time signature as specified in the abc file
- TUNEKEY – The musical key as specified in the abc file
- PART – the number of the tune part i.e. first, second, third part of the tune.
- BB_INTERVALS– intervals calculated using Breathnach's MIC system.

Figure 21 below shows rows 1 to 16 of the ABC corpus with pre-processed data. String comparison experiments were carried out directly on this data and the results stored in separate database tables.

#	ID	NAME	NOTES	TWOFOUR	INTERVALS	MEASURE	TUNEKEY	PART
1	20297	Annaghbeg	FFEDEFEEFEDCBABAFAFDFEDEFEEF...	FFEDEFEEFEDCBABAFAFDFEDEFEEF...	0,-1,-1,1,1,-1,0,1,-1,-1,-1,6,-1,1,-1,-2,2,-2...	2/4	Dmaj	3
2	20298	Brendan Begley''s	BDEDBGBDBGAAADEDBGBDBGGGBDE...	BEDGBBGAAEDGBBGGBEDGBBGGGGA...	-5,1,-1,5,-2,2,-5,5,-2,1,0,0,-4,1,-1,5,-2,2,-...	6/8	Gmaj	1
3	20299	Ballydesmond #1	AAAGEFGEAAAGADDCAGAGEFGFGA...	AAAGEFGEAAAGADDCAGAGEFGFGAB...	0,0,-1,-2,1,1,-2,3,0,0,-1,1,-4,0,-1,5,-1,1,-1...	2/4	Dmix	1
4	20300	Ballydesmond #1	EDDCDEBCCEDCDEAAGEDDCDEBCD...	EDDCDEBCCEDCDEAAGEDDCDEBCDC...	-1,0,-1,1,1,4,-6,0,2,-1,-1,1,1,3,0,-1,-2,-1,0...	2/4	Dmix	2
5	20301	Ballydesmond #2	EAABCCDEGGGGAGEEDEAGABCCDEF...	EAABCCDEGGGGAGEEDEAGABCCDEF...	3,0,1,-6,0,1,1,2,0,0,0,1,-1,-2,0,-1,1,3,-1,1,...	2/4	Ador	1
6	20302	Ballydesmond #2	AAABAAGEFGFGAGEDFDEAAGAGEFG...	AAABAAGEFGFGAGEDFDEAAGAGEFG...	0,0,1,-1,0,-1,-2,1,1,-1,1,1,-1,-2,-1,2,-2,1,3...	2/4	Ador	2
7	20303	Ballydesmond #3	BCCBCBABAGABCDBEDGGFGEAGGED...	BCCBCBABAGABCDBEDGGFGEAGGED...	-6,0,6,-6,6,-1,1,-1,-1,1,1,-6,1,5,-4,-1,3,0,-...	2/4	Ador	1
8	20304	Ballydesmond #3	EAAGEDGGFGEAAGAGGFGEAGAGED...	EAAGEDGGFGEAAGAGGFGEAGAGEDB...	3,0,-1,-2,-1,3,0,-1,1,-2,3,0,-1,1,-1,0,-1,1,-...	2/4	Ador	2
9	20305	Bill the Weaver''s	BDBEFABBABADBBEEFGAAGFEDBEEE...	BBEABABDBEEGAGFDBEEABABDAADF...	-5,5,-4,1,2,1,0,-1,1,-1,-4,5,0,-4,0,1,1,1,0,-...	6/8	Edor	1
10	20306	Bill the Weaver''s	GGEFEDEEBBAFBABEFGAAGFEDGFEF...	GEFDEBBFBBEGAGFDGEFDEBDBADDF...	0,-2,1,-1,-1,1,0,4,0,-1,-2,3,-1,1,-4,1,1,1,0,...	6/8	Edor	2
11	20307	Bill the Weaver''s	EEEEGEDEFGEEGEDEGEEFGAAGFEDE...	EEEEDFGEGDEEEGAGFDEEEDEGEFGG...	0,0,0,2,-2,-1,1,1,1,-2,0,2,-2,-1,1,2,-2,0,1,1...	6/8	Edor	3
12	20308	Bridgie Con Matt''s	DFGBBACBGGFGBBCDBADFDFGBBCB...	DFGBBACBGGFGBBCDBADFDFGBBCB...	2,1,2,0,-1,-5,6,-2,0,-1,1,2,0,-6,1,5,-1,-4,2,...	2/4	Gmaj	1
13	20309	Bridgie Con Matt''s	BEEFEDEFAGGBEFEDBADFDABEEFED...	BEEFEDEFAGGBEFEDBADFDABEEFED...	-4,0,1,-1,-1,1,1,2,-1,0,2,-4,1,-1,-1,5,-1,-4,...	2/4	Gmaj	2
14	20310	Callaghan''s	BCDEGFGEDBAGABCDBGEDEGABABC...	BEGEDGADBDEBADEFGBCDFDBEFAG...	-6,1,1,2,-1,1,-2,-1,5,-1,-1,1,1,-6,1,5,-2,-2,...	4/4	Gmaj	1
15	20311	Cobbler [The]	FGFEFBAFEFEDBBBCBABEFEDEFABA...	FGFEFBAFEFEDBBBCBABEFEDEFABA...	1,-1,-1,1,3,-1,-2,-1,1,-1,-1,5,0,0,-6,6,-1,1,...	2/4	Bdor	3
16	20312	Con Thadhgo''s	AFEFGFEFGFEEDEAFEFGFEEDDDDBA...	AEFFEGFEDAFFGEEDDBAEFEEEDBAE...	-2,-1,1,1,-1,-1,1,1,-1,-1,0,-1,1,3,-2,-1,1,1,...	6/8	Dmaj	1

Figure 21: ABC Corpus database rows 1 to 16 inclusive. Source: Author

6.2.3 Experiment Framework

Two separate frameworks were used to carry out experiments, a Java framework and a C Sharp (C#) framework. The description of each experiment indicates whether the Java or Microsoft C# dotNet platform was used to complete it.

6.2.3.1 Java Framework

A desktop Java application was created using the Netbeans IDE and the integrated Derby database server (Sun Microsystems 2010). This application provided a mechanism for iterating through rows of ABC data, performing string distance operations on pairs of tune parts and storing the results.

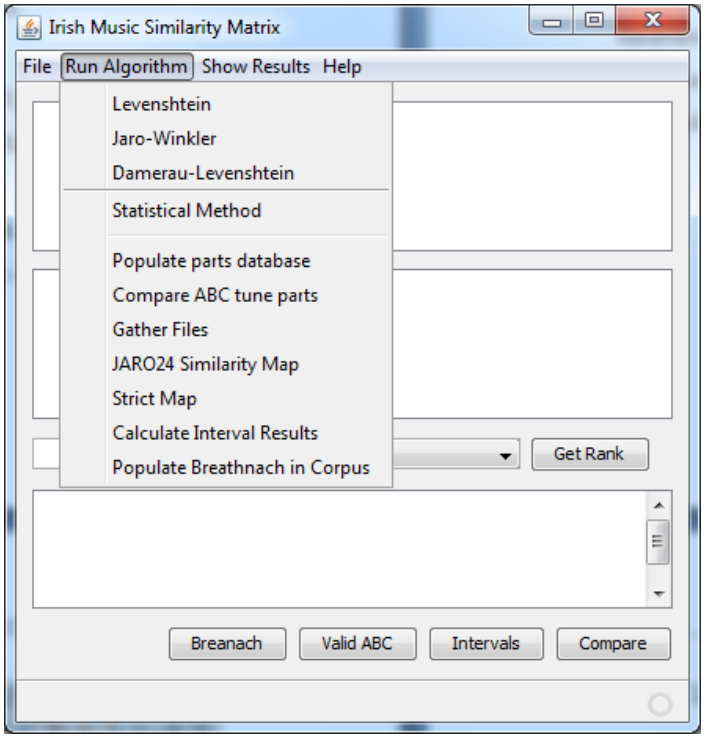


Figure 22: Desktop Java application framework for running experiments. Source: Author

6.2.3.2 C Sharp Framework

In order to construct the similarity matrix the Microsoft platform was used. This was primarily for performance issues identified while using the Derby database server but also to take advantage of MS SQL 2008's ability to use the Common Language Runtime (CLR) to create custom functions for use in SQL queries. For example, a

complex string distance algorithm could be converted into a SQL function and used directly on a database column as follows;

Table 26: MS SQL 2008 query using a custom function

```
SELECT      ID,      [Database].[dbo].[Jaro-Winkler](NOTES,
'ABCDEFG') from [Database].[dbo].[Table]
```

The SQL query in Table 26 returns the ID of each database row with the result of a Jaro-Winkler comparison between the string 'ABCDEFG' and every row in the entire NOTES column.

In addition, SimMetrics, a library of string distance algorithms programmed in the C# language, was already available containing implementations of the Levenshtein and Jaro-Winkler algorithms. MS SQL 2008's ranking functions were also taken advantage of to perform ranking experiments.

The platform used to construct the Similarity Matrix was as follows;

- SimMetrics C# String Distance Library
- Microsoft Visual Studio 2010 Professional Edition
- MS SQL Database Server 2008 Developer Edition

6.2.4 Levenshtein Experiments

The Levenshtein string comparison experiment was carried out on the Java platform and involved iterating over a number of tune parts and comparing them with a subset of the remaining rows. About 1,840 tune parts were compared against each other resulting in 3,386,248 comparisons. Figure 23 shows how the results were stored in a relational database.

SQL Command 1 | SQL Command 2

Connection: jdbc:derby://localhost:1527/ABC [padr...

```
1  select ID, TUNE_A_ID, TUNE_B_ID, LEVEN, LEVEN24 from APP.RESULTS
```

select ID, TUNE_A_ID, TUN

Page Size: 20 | Total Rows: 3386248 | Page: 1 of 169,313 | Matching Rows:

#	ID	TUNE_A_ID	TUNE_B_ID	LEVEN	LEVEN24
1	2368448	8353	8354	0.7051282051282052	0.7948717948717948
2	2368449	8353	8355	0.7435897435897436	0.7435897435897436
3	2368450	8353	8356	0.6410256410256411	0.6410256410256411
4	2368451	8353	8357	0.7564102564102564	0.7564102564102564
5	2368452	8353	8358	0.7564102564102564	0.7564102564102564
6	2368453	8353	8359	0.7307692307692307	0.7307692307692307
7	2368454	8353	8360	0.782051282051282	0.782051282051282
8	2368455	8353	8361	0.6025641025641025	0.7051282051282052
9	2368456	8353	8362	0.6153846153846154	0.717948717948718
10	2368457	8353	8363	0.6153846153846154	0.717948717948718
11	2368458	8353	8364	0.717948717948718	0.717948717948718
12	2368459	8353	8365	0.6666666666666666	0.6666666666666666
13	2368460	8353	8366	0.717948717948718	0.7435897435897436
14	2368461	8353	8367	0.6666666666666666	0.6666666666666666
15	2368462	8353	8368	0.5897435897435898	0.6666666666666666
16	2368463	8353	8369	0.6025641025641025	0.6794871794871795
17	2368464	8353	8370	0.7051282051282052	0.7435897435897436
18	2368465	8353	8371	0.6282051282051282	0.6282051282051282
19	2368466	8353	8372	0.6538461538461539	0.6538461538461539
20	2368467	8353	8373	0.6282051282051282	0.6282051282051282

Figure 23: Levenshtein comparison results. Source: Author

In this case the tune part with TUNE_A_ID 8353 was compared with the tune parts with TUNE_B_ID's 8354 to 8373 inclusive. Results of comparisons between the two NOTES columns are recorded in the LEVEN column while results of the comparison between the 2/4 versions of the tune parts are stored in the LEVEN24 column.

In order to plot the distribution of comparisons, the frequency of each result was obtained i.e. how many comparisons resulted in 0.01, how many resulted in 0.02 continuing to 0.99 and finally 1.0. This data was obtained for comparisons on both types of data (the original tune part and the 2/4 version) and plotted in Figure 24 and Figure 25.

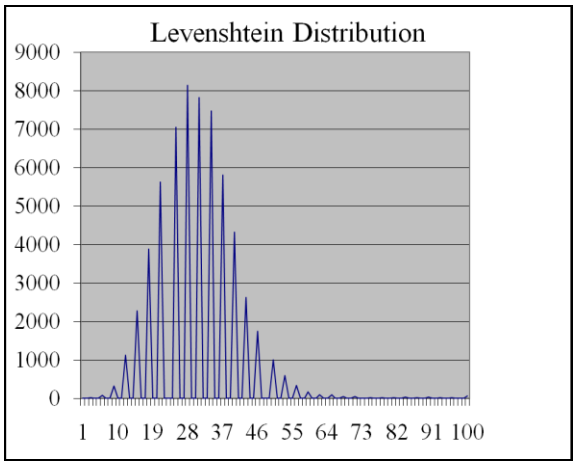


Figure 24: Levenshtein distribution

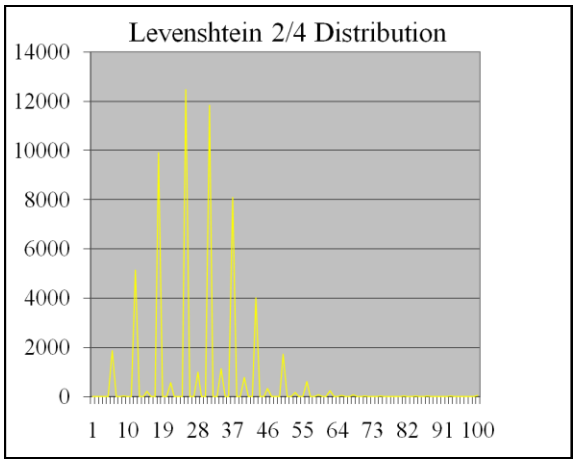


Figure 25: Levenshtein 2/4 distribution

As Figure 24 and Figure 25 show, the shapes of both distributions are almost identical. Both distributions show an off centre bell curve with the majority of the results in the 12% to 64% area. Similar to the distribution Müllensiefen & Frieler found in Figure 26 (Mullensiefen & Frieler 2007, p.196) the distribution of a Levenshtein comparison of the whole corpus looks much like an off centre normal distribution. In this experiment results below 12% and above 64% were very rare.

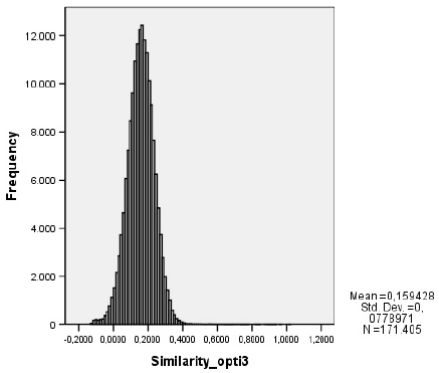


Figure 26: Frequency distribution by M&F of all melodies in their database. Source: (Mullensiefen & Frieler 2007, p.196)

6.2.5 Jaro-Winkler Experiments

The Jaro-Winkler experiments were carried out in parallel with the Levenshtein experiments as both experiments required iteration over the same data.

As with the Levenshtein experiments, the frequency of each result was determined in order to calculate the Jaro-Winkler distribution of results. This was performed for both sets of data (the original tune part and the 2/4 version) and plotted on a line graph.

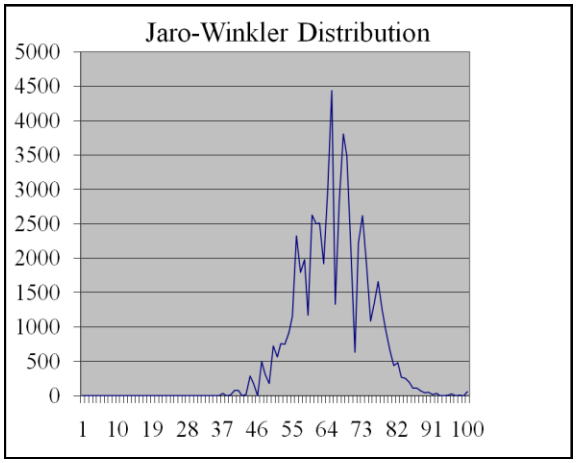


Figure 27: Jaro-Winkler distribution

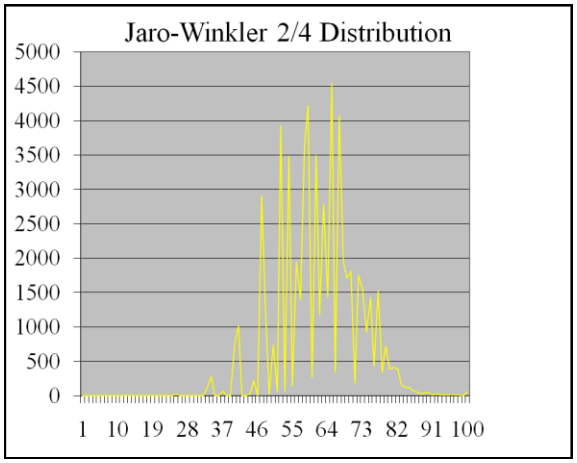


Figure 28: Jaro-Winkler 2/4 distribution

In the same manner as the Levenshtein result, the Jaro-Winkler distribution graph showed that the bulk of the results were between a certain range (37% - 90%). Once again the shape of the graph resembled an off centre normal distribution curve. Results above 90% were very rare with results below 37% being virtually non-existent.

Processing 2/4 data compared to unaltered data with the Jaro-Winkler algorithm results in the distortion of the normal distribution curve apparent in Figure 27.

6.2.6 Lemström Semex Interval Experiments

The Lemström Semex algorithm is a very efficient, transposition invariant algorithm capable of identifying sub-sequences of note patterns in large music databases. Preliminary tests on a subset of data showed that the Semex algorithm was capable of identifying the same melody in different musical keys. Similar to the Levenshtein algorithm an integer is returned which signifies the edit distance between a sub-sequence and a larger string. It was found that it was possible to normalise the result by dividing this distance by the length of the shorter search string. The Semex algorithm has been shown to be applicable in many environments including searching polyphonic music (Dovey 2001), fault tolerant music identification (Clausen & Kurth 2002), a web based music retrieval system (Rho & Hwang 2004), matching melody directly from audio (Mazzoni & RB Dannenberg 2001) and various other

environments. For this reason it was decided that this experiment would not test how effective the algorithm is at identifying similarity in the context of music but instead if it would be possible to use this algorithm to construct a Music Similarity Matrix (MSM) and if so, what resources would be needed.

This experiment was carried out on the Java platform. As with the previous experiments, the Semex algorithm was used to compare tune pairs of the original melody and the measure invariant 2/4 version of the melody. The un-normalised edit distance was stored for both comparison types.

The experiment was run on the laptop described in Section 6.2 on a corpus of 11,944 tune parts. This meant that 142,647,192 comparisons would need to be performed in order to complete the similarity matrix. The experiment was halted after five days and 49,908,185 comparisons when writing to the hard disc became extremely slow, effectively rendering the experiment impossible to complete within the available resources.

It was possible to draw some conclusions from this experiment even though it was not completed as measurable data resulted. Over the course of the experiment the following was observed;

- Approximately one third of the matrix was completed in five days meaning that a full matrix could be completed in just over two weeks.
- 50 million database rows used 2.5 gigabytes of hard disk space. The total amount of hard disk space a complete matrix would require is approximately 7.5 gigabytes.
- Querying the database of results was very slow, taking an average of over eight minutes to complete even simple queries as in Figure 29.

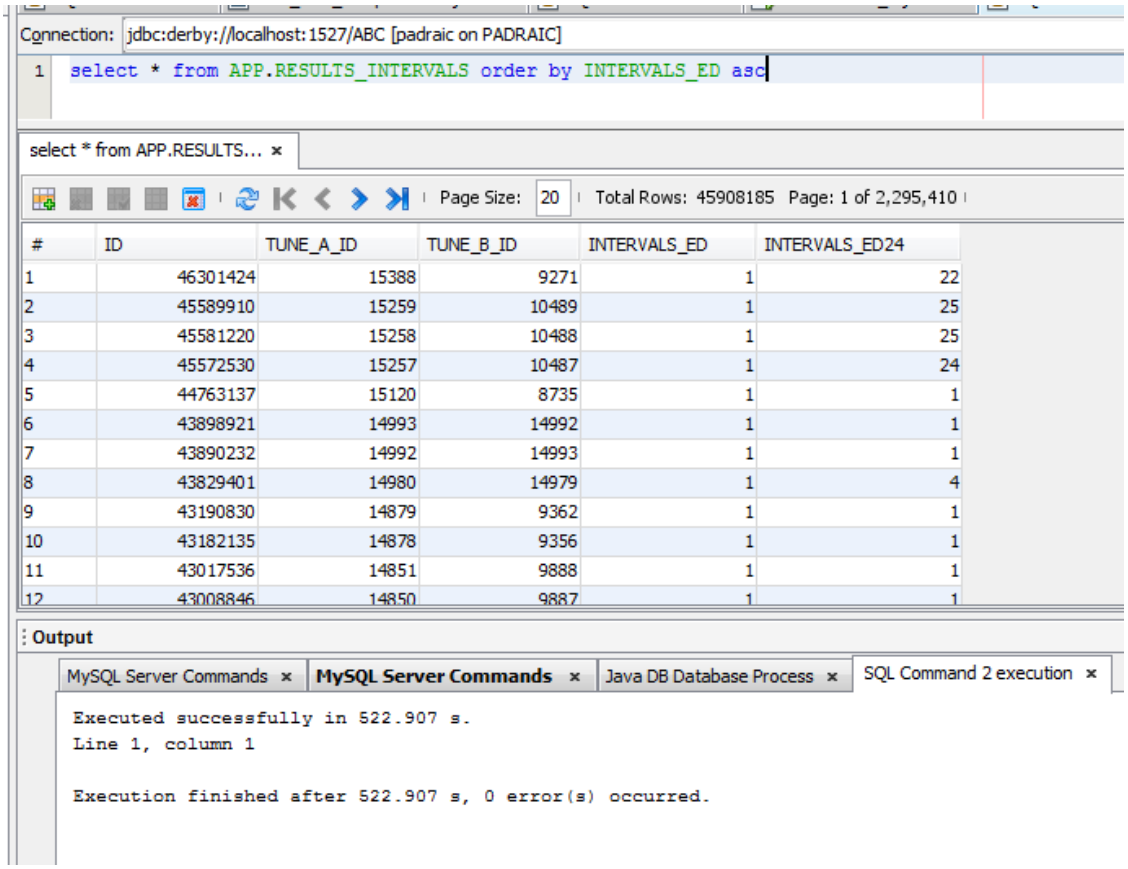


Figure 29: Simple SQL query on interval data taking 523 seconds.

The following conclusions were drawn from the experiment;

- Greater computing resources are required in order to complete a music similarity matrix using the methods in this experiment.
- Database and SQL query optimisations would need to be performed.
- Only comparisons with results within a certain threshold (to be determined) should be stored in order to minimise hard disk usage.
- Constructing an extremely large database was fruitless unless it could be reduced and analysed in a meaningful way.
- Alternative solutions to constructing the matrix should be investigated and considered.

6.2.7 Melodic Indexing Code experiments

The purpose of this experiment was to computerise Breandán Breathnach's Melodic Indexing System as described in Section 2.2. It was carried out on the Java platform. A series of steps were planned in order to complete this experiment as follows;

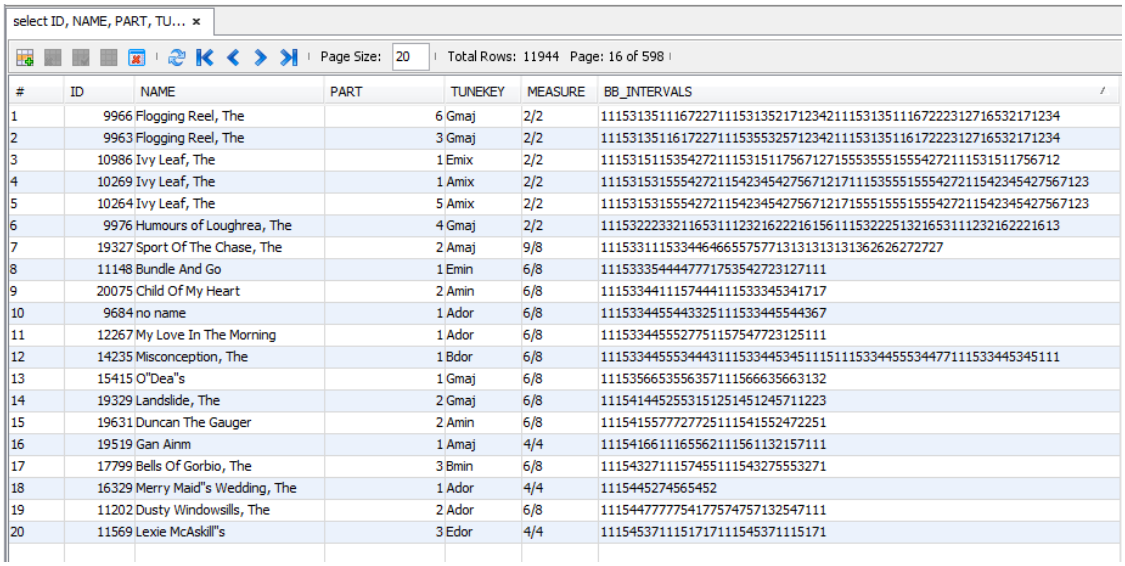
- A Java method would be developed in order to calculate intervals with respect to a fundamental note.
- All of the corpus would be converted to Melodic Index Codes (MIC) with resulting MIC codes stored with the original tune part.
- Two different sorting methods, numeric and alphabetic would be tested and evaluated.

Once preliminary testing was completed, bugs had been identified and corrected the experiment was run using the Java framework developed in Section 6.2.3. The experiment completed in less than 30 minutes without issue and the testing and evaluation of sorting methods began.

Figure 30: The Munster Lass jig stored in the Breathnach Melodic Indexing System. Source: Author

Breathnach stored index cards by tune type in numerical order. Jigs were stored separately from reels, hornpipes, slides and polkas as can be seen in Figure 30.

A visual check of the results confirmed that the system is transposition invariant, correctly matching The Ivy Leaf reel in two keys, E mixolydian and A mixolydian, rows 3 & 4 of Figure 31.



select ID, NAME, PART, TU... ×

Page Size: 20 | Total Rows: 11944 Page: 16 of 598

#	ID	NAME	PART	TUNEKEY	MEASURE	BB_INTERVALS
1	9966	Flogging Reel, The	6	Gmaj	2/2	11153135111672271115313521712342111531351116722231271653217 1234
2	9963	Flogging Reel, The	3	Gmaj	2/2	11153135116172271115355325712342111531351161722231271653217 1234
3	10986	Ivy Leaf, The	1	Emix	2/2	11153151153542721115315117567127155535551555427211153151175 6712
4	10269	Ivy Leaf, The	1	Amix	2/2	11153153155542721154234542756712171115355515554272115423454275 67123
5	10264	Ivy Leaf, The	5	Amix	2/2	11153153155542721154234542756712171555155515554272115423454275 67123
6	9976	Humours of Loughrea, The	4	Gmaj	2/2	11153222332116531112321622216156111532225132165311123216222161 3
7	19327	Sport Of The Chase, The	2	Amaj	9/8	11153311153344646655757713131313131362626272727
8	11148	Bundle And Go	1	Emin	6/8	11153335444477717535427231271 11
9	20075	Child Of My Heart	2	Amin	6/8	11153344111574441115333453417 17
10	9684	no name	1	Ador	6/8	11153344554433251115334455443 67
11	12267	My Love In The Morning	1	Ador	6/8	11153344555277511575477231251 11
12	14235	Misconception, The	1	Bdor	6/8	11153344555344431115334453451115111533445553447711153344534 5111
13	15415	O"Dea"s	1	Gmaj	6/8	11153566535563571115666356631 32
14	19329	Landslide, The	2	Gmaj	6/8	11154144255315125145124571122 3
15	19631	Duncan The Gauger	2	Amin	6/8	11154155777277251115415524722 51
16	19519	Gan Ainm	1	Amaj	4/4	11154166111655621115611321571 11
17	17799	Bells Of Gorbio, The	3	Bmin	6/8	11154327111574551115432755532 71
18	16329	Merry Maid"s Wedding, The	1	Ador	4/4	1115445274565452
19	11202	Dusty Windowsills, The	2	Ador	6/8	11154477777541775747571325471 11
20	11569	Lexie McAskill"s	3	Edor	4/4	11154537111517171115453711151 71

Figure 31: Computerised Melodic Indexing System

Figure 31 shows that comparisons between a range of time signatures and type of tune are possible under the system. For example, row 16, a reel without a name in the key of A Major was found to be similar to an A Minor jig called the Drunken Gauger (row 15) and another jig called the Bells of Gorbio in B Minor (row 17).

6.3 Evaluation

6.3.1 Survey of experts and non-experts

In order to test whether a computer algorithm could accurately identify similar or different tune parts, an online survey was conducted. Participants were divided into two groups, those that could be considered experts in Irish traditional music and those that had no interest or experience in Irish traditional music.

6.3.2 Choosing tune part pairs to test

Pairs were selected based on the following criteria;

- Normalised Levenshtein score
- Jaro-Winkler score
- Breathnach Melodic Index Proximity

Levenshtein scores had to be normalised in order to take account of comparing different length strings of musical notes. Two strings of notes, ten and twenty characters long respectively have an edit distance of at least ten. An edit distance of ten is considerably more significant if the strings are 1 and 11 characters long than if they are 101 and 111 characters long. In order to normalise the Levenshtein scores the following formula was used;

$$\frac{ed}{len(s1) + len(s2)}$$

Where ed is the Levenshtein edit distance, len(s1) is the number of characters in string one and len(s2) is the number of characters in string two.

Table 27: List of tune pairs selected for the survey

Pair	ID	Tune A	Part	Leven	Leven 24	Jaro	Jaro 24
1a	10101	Jenny's Chickens	1	0.19	0.38	0.57	0.59
1b	11475	Sean sa Cheo	1				
2a	8544	All the world loves me	1	0.56	0.68	0.72	0.62
2b	8749	Lackeys	1				
2a	8545	All the world loves me	2	0.53	0.36	0.63	0.78
2b	8750	Lackeys	2				
3a	14736	A maid that dare not tell	2	0.31	0.5	0.75	0.65
3b	8542	All around the room	1				
4a	11324	The Musical Priest	1	0.66	0.82	0.88	0.91
4b	12006	North Brig O'Edinburgh	1				
5a	9972	Jenny picking cockles	1	0.47	0.44	0.70	0.67
5b	10944	Repeal of the Union	1				
6a	13546	Whisky makes you a lunatic	1	0.06	0.00	0.29	0.00
6b	18031	En Dro	2				
7a	14845	Humours of Tulla	2	0.06	0.06	0.37	0.25
7b	16191	Farewell to Stromness	2				
8a	16195	Willie Davie	2	0.22	0.19	0.52	0.48
8b	17356	Hangmans	3				
9a	18579	Oliver Jack	1	0.09	0.19	0.38	0.48
9b	8618	Down the Gort Road	2				
10a	10101	Jenny's Chickens 2/4	1	0.19	0.38	0.57	0.59
10b	11475	Sean sa Cheo 2/4	1				

6.3.3 How tune pairs were chosen

Table 27 shows the tune part pairs with their Levenshtein and Jaro-Winkler similarity scores for unaltered and 2/4 version of the musical notes.

6.3.3.1 Pairs 1 & 10

The same tune pair, *Jenny's Chickens* and *Sean sa Cheo* were used for pairs 1 and 10. Question 1 contained both tunes in 4/4 time signature and Question 10 contained both tunes in 2/4 time signature.

This example represents a weakness in the Levenshtein and Jaro-Winkler string distance algorithms' inability to compensate for vertically transposed melodies as they cannot perform key invariant comparisons. On the other hand, this pair ranked highly on the Breathnach Melodic Indexing System being 245 tunes apart in a corpus of 11944.

6.3.3.2 Pairs 2, 3, 4 and 5

Pairs 2 and 5 were chosen because they had relatively high Jaro-Winkler scores and average Levenshtein scores. Pair 3 was chosen because it had a relatively high Jaro-Winkler score but a low Levenshtein score. Pair 4 had a relatively high Levenshtein score and a high Jaro-Winkler score. All of these pairs were deemed to be similar according to at least one algorithm.

6.3.3.3 Pairs 6, 7, 8 and 9

Pairs 6, 7 and 9 were chosen because they had low Levenshtein and Jaro-Winkler scores. Pair 8 was chosen because it had an average Jaro-Winkler score but a low Levenshtein score. All of these pairs were deemed to be dissimilar according to at least one algorithm.

6.3.4 Question order randomisation

In order to ensure that the order did not bias the survey results a list of tune pairs was prepared. This list was then randomised and the online survey constructed accordingly. The names of the tunes were not available to the participants and the ABC tune part data was converted to audio by computer instead of recording a musician playing a

version of each tune part. This was done in order to prevent bias due to style of playing or instrument choice.

Table 28: List of tune pairs selected for the survey

Pair No.		Method	Order
1	Similar	Breathnach	9
2	Similar	Levenshtein & Jaro-Winkler	3
3	Similar	Levenshtein & Jaro-Winkler	10
4	Similar	Levenshtein & Jaro-Winkler	5
5	Similar	Levenshtein & Jaro-Winkler	4
6	Different	Levenshtein & Jaro-Winkler	6
7	Different	Levenshtein & Jaro-Winkler	2
8	Different	Levenshtein & Jaro-Winkler	7
9	Different	Levenshtein & Jaro-Winkler	1
10	Similar	Breathnach	8

According to the methods used to determine similarity and dissimilarity the survey contained the following tests.

Question Number	Computer Determination	Pair Number
1	Different	9
2	Different	7
3	Similar	2
4	Similar	5
5	Similar	4
6	Different	6
7	Different	8
8	Similar	10
9	Similar	1
10	Similar	3

6.3.5 Choosing experts

In order to ensure that a representative result from the survey was returned, great care was taken when choosing a panel of experts. For the purposes of the survey experts were distinguished from non-experts and a minimum criteria was established before a candidate was considered to be an expert or a non-expert in the field of Irish music. A minimum criteria was also formulated in order to identify non-experts. The minimum criteria were as follows;

An expert must;

- Play a musical instrument that Irish music would normally be played on.
- Have played Irish traditional music for at least 15 years.

A non-expert must;

- Not play any musical instrument.
- Not listen to Irish traditional music regularly

Experts and non-experts alike could;

- Be of any nationality
- Be of any gender orientation
- Be of any age
- Should not be tone deaf

Lists of experts and non-experts are presented in Appendix A.

6.3.6 Experts results

A panel of experts were asked to choose whether tune parts were similar or different using a Likert scale (Likert 1932). Each participant was presented with twenty audio clips grouped into ten pairs of tune parts. The expert was instructed to play each pair of clips as many times as necessary in order to make a decision. The Likert scale allowed each participant to choose one of five options. The choices given to each participant were as follows;

Table 29: Likert Scale Values

Likert Scale	Value
Very different	1
Different	2
I dont know	3
Similar	4
Very similar	5

The corresponding values for the Likert responses given by each expert participating in the survey are shown in Table 30.

Table 30: Responses from participants that are experts in Irish traditional music

Name	Q 1	Q 2	Q 3	Q 4	Q 5	Q 6	Q 7	Q 8	Q 9	Q 10
Hauke Steinberg	4	2	5	2	5	2	5	1	4	4
David Morrissey	4	4	3	4	5	4	5	2	2	2
Martin Preshaw	2	4	4	5	1	1	5	1	2	1
Daragh O'Reilly	2	4	4	4	5	2	5	1	4	2
Jose Manuel Fernandez Mateos	1	1	2	1	4	1	5	1	4	1
Deirdre Smyth	2	5	5	5	4	4	5	2	5	4
Damian Werner	4	5	4	5	1	1	4	1	5	4
Paulo McNevin	1	1	1	1	5	1	5	1	1	1
Ray Dempsey	1	2	4	2	5	1	5	2	5	2
Terry McGee	1	1	1	1	2	1	4	1	4	1
Pádhraic ó Súilleabhán	1	4	2	4	4	2	4	4	4	2
Treasa Lavin	2	2	4	2	1	1	5	1	5	2
Joe Brennan	2	4	4	5	5	2	5	1	5	3
Pauline Burke	2	4	5	4	4	1	4	1	4	1
Sara Cory	4	2	5	4	2	1	5	1	4	2

Table 31 shows how the experts voted.

Table 31: Results of experts choices

Question No.	Similar	Different	Not sure	Conclusion
Question 1	4	11	0	Different
Question 2	8	7	0	Similar
Question 3	10	4	1	Similar
Question 4	9	6	0	Similar
Question 5	10	5	0	Similar
Question 6	2	13	0	Different
Question 7	15	0	0	Similar
Question 8	1	14	0	Different

Question 9	12	3	0	Similar
Question 10	3	11	1	Different
Totals	74	74	2	150

6.3.6.1 Analysis of the experts responses

Most questions resulted in experts voting by a majority of over 10 votes to 5 except in two cases. In question 2 and 4 the experts voted by a majority of 8 to 7 and 9 to 6 respectively. The outcomes of question 2 and 4 may be inconclusive as the experts seem to be unsure of their collective decisions.

6.3.7 Non-experts results

Survey participants with no musical experience were given the same survey as the experts under exactly the same conditions. Their responses are given in the table below.

Table 32: Responses from participants with no experience of Irish traditional music

Name	Q 1	Q 2	Q 3	Q 4	Q 5	Q 6	Q 7	Q 8	Q 9	Q 10
Corinne Kingston Bageard	4	5	4	2	2	2	5	2	5	2
Diarmuid Cooke	1	1	4	2	4	4	5	1	5	2
Brian Duggan	2	4	5	4	2	2	5	1	4	2
Martin Hughes	2	2	5	4	2	2	5	2	4	2
Joe Phelan	2	5	4	1	1	2	5	1	4	4
John Golden	2	4	5	2	1	2	5	2	4	4
Patrick Crowe	1	2	2	1	2	1	2	1	2	1
John Breen	1	2	4	5	1	1	5	1	5	2
Caroline Bemingham	2	2	5	5	1	1	5	1	5	2
Mark Bussell	2	4	5	4	4	1	5	2	4	5
Enora Senlanne	2	4	4	4	4	1	4	1	5	2
Richard Kinser	2	4	4	4	4	2	5	2	4	2
Terry Lavin	2	1	2	5	1	1	5	1	5	1
Clare Basset	4	4	4	5	4	2	5	3	5	2
Louisa Murphy	4	2	2	5	4	1	5	1	4	2

Table 33: Results of non-experts choices

Question No.	Similar	Different	Not sure	Conclusion
Question 1	3	12	0	Different
Question 2	8	7	0	Similar
Question 3	12	3	0	Similar
Question 4	10	5	0	Similar

Question 5	6	9	0	Different
Question 6	1	14	0	Different
Question 7	14	1	0	Similar
Question 8	0	14	1	Different
Question 9	14	1	0	Similar
Question 10	3	12	0	Different
Totals	71	78	1	150

6.3.7.1 Analysis of the non-experts responses

Most questions resulted in the non-experts voting by a majority of over 10 votes to 5 except in two cases. In question 2 and 5 the experts voted by a majority of 8 to 7 and 6 to 9 respectively. The outcomes of question 2 and 5 may be inconclusive as the non-experts seem to be unsure of their collective decisions.

6.3.8 Experts vs. non-experts

Interestingly, the expert and non-expert participants agree on all questions apart from one pair, question 5.

Question No.	**Experts**	**Non-experts**
Question 1	Different	Different
Question 2	Similar	Similar
Question 3	Similar	Similar
Question 4	Similar	Similar
Question 5	Similar	Different
Question 6	Different	Different
Question 7	Similar	Similar
Question 8	Different	Different
Question 9	Similar	Similar
Question 10	Different	Different

The numbers of votes for each tune pair were counted in order to calculate voting percentages for each question answered by both groups. For example, four experts out of fifteen voted that the tune pair in Question 1 were similar resulting in 27% of the vote and three out of fifteen non-experts voted that the tune pair in Question 1 were similar giving a vote of 20%. When the votes for both groups are plotted on a chart the results look remarkably similar.

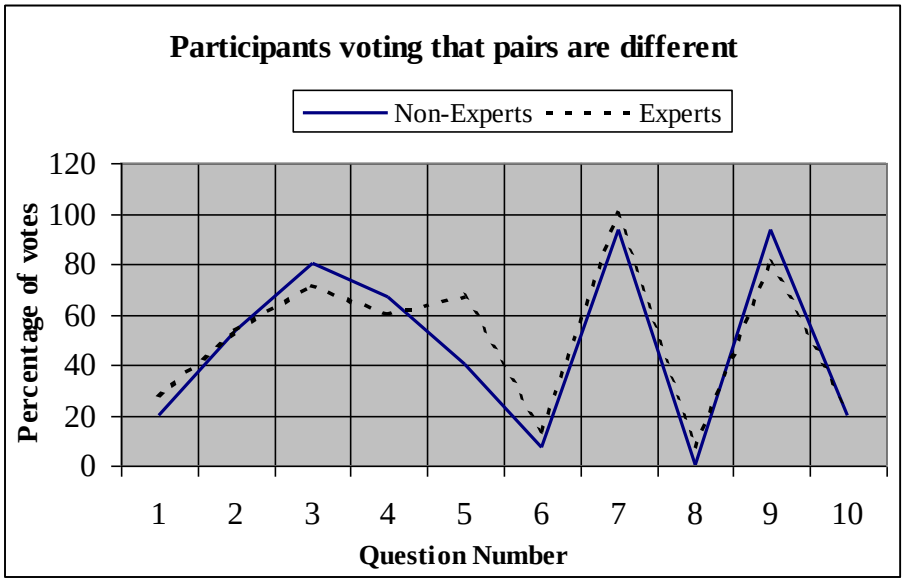


Figure 32: Experts vs. non-experts voting percentages

The following table shows how the computer algorithm chose similarities contrasted with those of the experts and non-experts.

Table 34: Computer algorithm vs expert vs non-expert choices

Question No.	Computer algorithm	Experts	Non-experts
1	Different	Different	Different
2	Different	Similar	Similar
3	Similar	Similar	Similar
4	Similar	Similar	Similar
5	Similar	Similar	Different
6	Different	Different	Different
7	Different	Similar	Similar
8	Similar	Different	Different
9	Similar	Similar	Similar
10	Similar	Different	Different

An analysis of these results suggests that experts and non-experts are likely to choose similarly. The one question where experts and non-experts differ is question 5 but this result may be classified as inconclusive because the voting is so close as to suggest that opinion was almost equally divided in both groups.

The tune pairs selected by the computer algorithm agreed with the experts at least 60% of the time. Question 2 was voted similar by a margin of 8 to 7 in both groups suggesting that opinion in humans was narrowly divided. The voting from both groups for questions 7, 8 and 10 suggests that the computer algorithm made a significant error selecting these pairs.

6.4 Constructing a Similarity Matrix for Irish Traditional Music

Using the process defined in Section 4.3 an experiment was designed in order to construct four similarity matrices. These matrices were constructed using the Jaro-Winkler algorithm, Parsons Code, Melodic Indexing System and the Combined Ranking System described in Section 4.3. Construction was carried out over four phases.

6.4.1 Phase 1 – Importing data and extending MS SQL 2008

The first phase involved importing the corpus of tunes from the Derby database server into the MS SQL 2008 database server. As this data had already been cleaned and processed numerous times in other experiments it made sense to use it for experiments on the Microsoft platform. Comparisons between both platforms may also be made possible in the future. This phase also involved extending the MS SQL 2008 database server by writing implementations of the Lemström Semex, Breathnach Melodic Indexing System, Parsons Code and a standard deviation function in the C# language. These implementations are available in Appendix D.

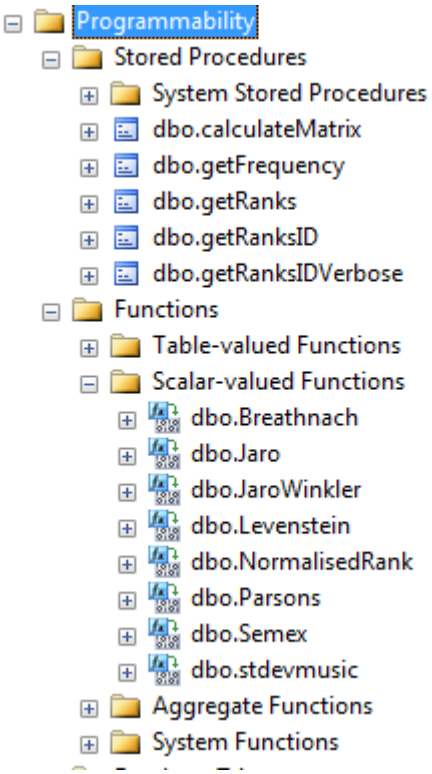


Figure 33: Stored Procedures and custom functions in MS SQL 2008

Figure 33 shows a screenshot of Microsoft Management Studio (the application used to administer MS SQL 2008). This screenshot shows how MS SQL Server 2008 has been extended by using custom stored procedures getRanks, getRanksID and custom functions Breathnach, Jaro-Winkler, Levenshtein, Parsons, stdevmusic and NormalisedRank.

6.4.2 Phase 2 - Testing custom function SQL queries

The purpose of creating custom functions using Microsoft Visual Studio 2010 Professional to extend MS SQL 2008 was to enable the use of string distance functions within SQL queries. Two Visual Studio projects were used; the first to extend the SimMetrics string distance library to include implementations of the Semex, Parsons, Breathnach MIC and improved Jaro-Winkler algorithms and the second to create a private dotnet assembly that could be imported into MS SQL 2008. In order to test if these custom functions worked as planned in MS SQL 2008 the following SQL query was executed.

Table 35: SQL query using a custom string distance function

```
select ID, NAME, NOTES, [Test].dbo.JaroWinkler(NOTES, 'ABCC') as
JW_Score
```

```
from Test.dbo.corpus
order by JW_Score desc
```

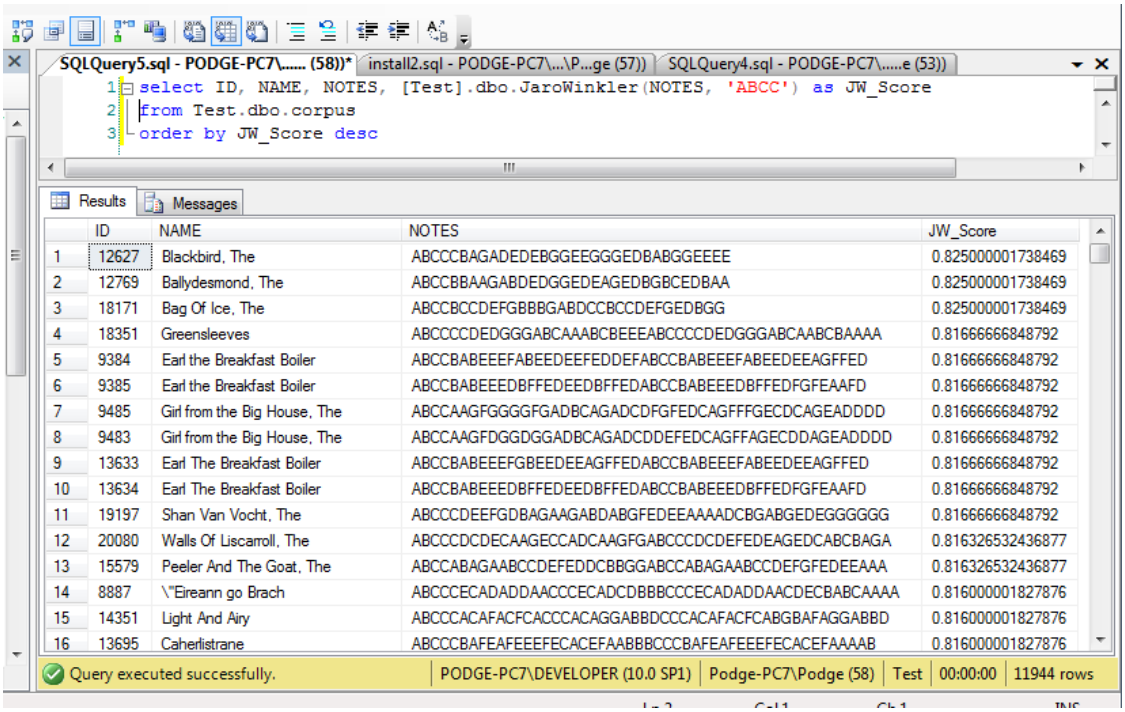


Figure 34: Result of a SQL query using a custom string distance function

Figure 34 shows the following columns returned by the SQL query;

- ID of the tune
- NAME of the tune
- NOTES of the tune
- JW_Score represents the similarity between the string 'ABCC' and each of the rows in the NOTES column in descending order

The bottom right of the screenshot shows that the corpus of 11944 rows was processed in less than 1 second. Table 36 shows how a more complex query was then executed. It compared the notes from the tune "The Humours of Tulla" to the corpus of 11944 tunes using the Jaro-Winkler, Levenshtein and Semex custom functions.

Table 36: Jaro-Winkler, Levenshtein and Semex SQL for the "Humours of Tulla"

```
select ID, NAME, NOTES, [Test].dbo.JaroWinkler(NOTES,
'GGDGEGDEGGBGAGEFGGDGEGDGEFGABCCBA') as JW_Score,
[Test].dbo.Levenstein(NOTES, 'GGDGEGDEGGBGAGEFGGDGEGDGEFGABCCBA') as
Leven_Score,
[Test].dbo.Semex('GGDGEGDEGGBGAGEFGGDGEGDGEFGABCCBA', NOTES) as
Semex_Score
from Test.dbo.corpus
```

```
order by Semex_Score desc
```

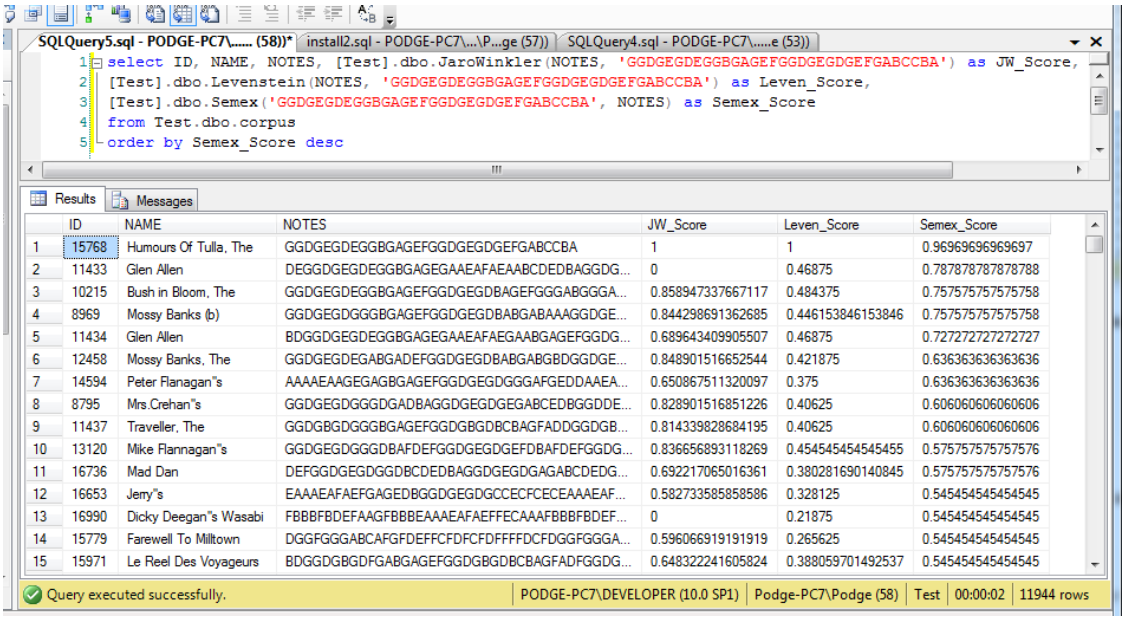


Figure 35: Jaro-Winkler, Levenshtein and Semex SQL query combined

Figure 35 shows how the SQL query in Table 36 was executed, comparing the notes of the "Humours of Tulla" against the whole corpus of tunes using the Levenshtein, Jaro-Winkler and Semex custom functions in just 2 seconds.

Custom functions that returned Parsons Code and Breathnach's Melodic Indexing Code were also created. These two functions return the Parsons Code and MIC Code rather than a similarity score between 0 and 1. In order to calculate how proximate two strings of notes are, their positions in the corpus must first be known. A custom function in MS SQL Server 2008 is only aware of the two strings of notes passed to it as arguments and not aware of the entire corpus of tunes. It was decided that it would be more appropriate to perform this type of calculation within a stored procedure that would have access to both custom string distance functions and the whole corpus. The following SQL example shows how it is possible to convert a whole corpus of tunes to Melodic Indexing Code and Parsons Code in a few seconds.

Table 37: SQL query to convert a corpus into MIC Code and Parsons Code

```
select ID, NAme, [Test].dbo.Breathnach(NOTES) as MIC,
[Test].dbo.Parsons(NOTES) as PIC
from Test.dbo.corpus
order by MIC asc
```

Figure 36 shows how the SQL query above converts a corpus of 11944 tunes into MIC Code and Parsons code in one second.

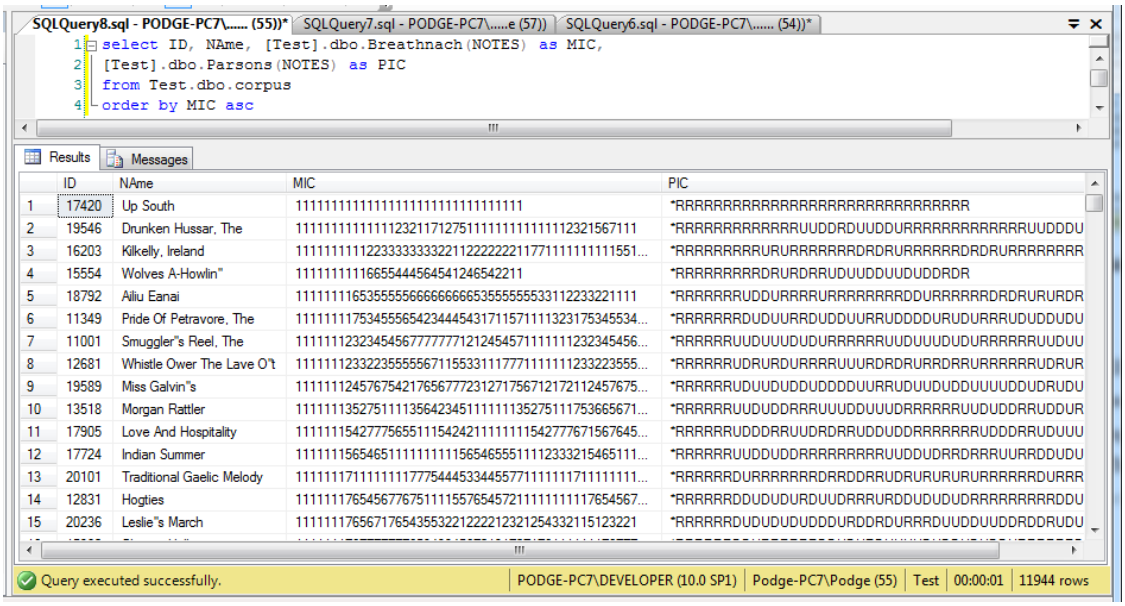


Figure 36: Corpus of tunes in MIC Code and Parsons Code

Temporary tables in stored procedures were used to dynamically create corpi so that positions of tunes within them could be ascertained. Once the position of the match was known it was possible to calculate proximity and distance from this match as described in Figure 6. Both MIC Code and Parsons Code were calculated in tandem as the systems are virtually identical (apart from the generated MIC and Parsons Code).

Following is code that creates a temporary table, calculates MIC and Parsons Code for an entire corpus of tunes, identifies the closest match using both Melodic Index Code and Parsons Code, calculates distance and then normalises the MIC and Parsons distances so that a score between 0 and 1 is returned.

Table 38: Code snippet that calculates and normalises MIC & Parsons Code ranks

```
-- Create temp table for Breathnach and Parsons Rank and populate it
      SELECT      ID,
      dbo.Breathnach(NOTES) as MIC,
      dbo.Parsons(NOTES) as PIC,
      row_number() over (order by dbo.Breathnach(NOTES) asc) as
rowID,
      (row_number() over (order by dbo.Breathnach(NOTES)))/1.0 as
MICScore,
      (row_number() over (order by dbo.Parsons(NOTES)))/1.0 as
PICScore
      into #TEMP
      from corpus
      order by MIC;
```

```
      -- Find nearest match for @notes - Breathnach
      Select top 1 @rowID = MICScore from #TEMP where
dbo.Breathnach(@notes) <= MIC order by MIC asc
      Select @MaxRank = MAX(MICScore) from #TEMP

      -- Find nearest match for @notes - Parsons
      Select top 1 @prowID = PICScore from #TEMP where
dbo.Parsons(@notes) <= PIC order by PIC asc
      Select @PMaxRank = MAX(PICScore) from #TEMP

      Update #TEMP set MICScore = 1-((abs(MICScore -
@rowID))/@MaxRank)
      Update    #TEMP    set    PICScore    =    1-((abs(PICScore    -
@prowID))/@PMaxRank)
```

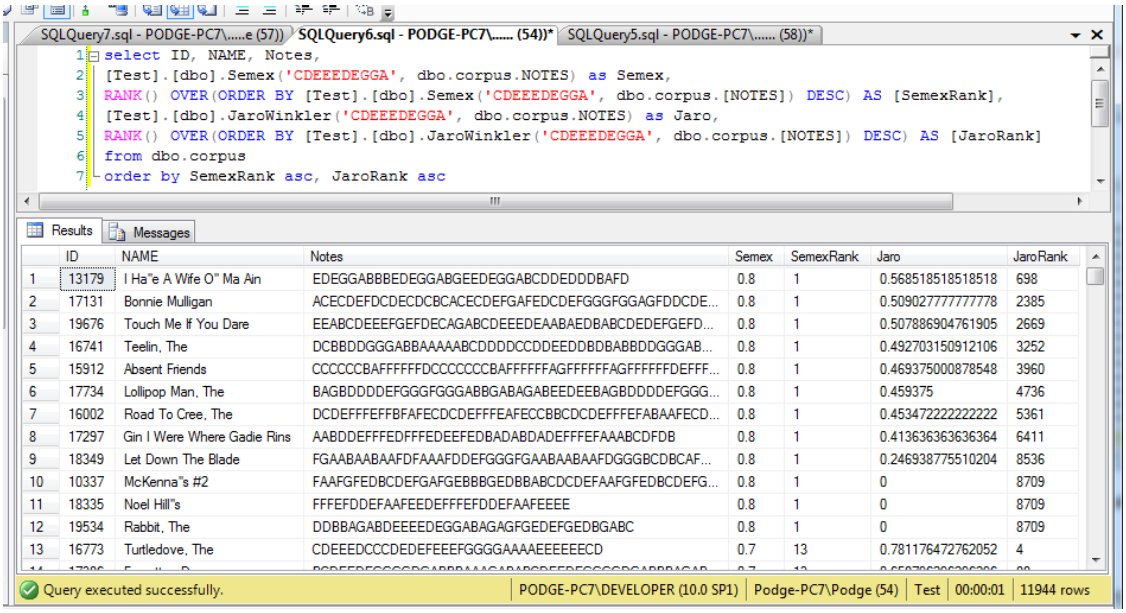
The complete stored procedure is available in Appendix D.

6.4.3 A Combined Ranking System

MS SQL 2008 supports four ranking functions, one of which, RANK() was used to generate ranks for results returned by string distance functions. Table 39 shows a SQL query that utilises the RANK() function in conjunction with the Jaro-Winkler and Semex string distance custom functions.

Table 39: SQL query for Semex & Jaro-Winkler scores with ranks

```
select ID, NAME, Notes,
[Test].[dbo].Semex('CDEEEDEGGA', dbo.corpus.NOTES) as Semex,
RANK() OVER(ORDER BY [Test].[dbo].Semex('CDEEEDEGGA',
dbo.corpus.[NOTES]) DESC) AS [SemexRank],
[Test].[dbo].JaroWinkler('CDEEEDEGGA', dbo.corpus.NOTES) as Jaro,
RANK() OVER(ORDER BY [Test].[dbo].JaroWinkler('CDEEEDEGGA',
dbo.corpus.[NOTES]) DESC) AS [JaroRank]
from dbo.corpus
order by SemexRank asc, JaroRank asc
```



```
select ID, NAME, Notes,
[Test].[dbo].Semex('CDEEEDEGGA', dbo.corpus.NOTES) as Semex,
RANK() OVER(ORDER BY [Test].[dbo].Semex('CDEEEDEGGA', dbo.corpus.[NOTES]) DESC) AS [SemexRank],
[Test].[dbo].JaroWinkler('CDEEEDEGGA', dbo.corpus.NOTES) as Jaro,
RANK() OVER(ORDER BY [Test].[dbo].JaroWinkler('CDEEEDEGGA', dbo.corpus.[NOTES]) DESC) AS [JaroRank]
from dbo.corpus
order by SemexRank asc, JaroRank asc
```

	ID	NAME	Notes	Semex	SemexRank	Jaro	JaroRank
1	13179	I Ha"e A Wife O" Ma Ain	EDEGGABBBEDEGGABGEEDEGGABCDDEDDDBAFD	0.8	1	0.568518518518518	698
2	17131	Bonnie Mulligan	ACECDEFDCDECDCBCACECDEFGAFEDCDEFGGGFGGAGFDDCDE...	0.8	1	0.509027777777778	2385
3	19676	Touch Me If You Dare	EEABCDEEEFGEFDECAGABCDEEEDEAABAEDBABCDEDEFGEFD...	0.8	1	0.507886904761905	2669
4	16741	Teelin, The	DCBBDDGGGABBAAAAABCDDDDCCDDEEDDBDBABBDDGGGAB...	0.8	1	0.492703150912106	3252
5	15912	Absent Friends	CCCCCCBAFFFFFFDCCCCCCCBAFFFFFFAGFFFFFFAGFFFFFFDEFFF...	0.8	1	0.469375000878548	3960
6	17734	Lollipop Man, The	BAGBDDDDEFGGGFGGGABBGABAGABEEDEEBAGBDDDDEFGGG...	0.8	1	0.459375	4736
7	16002	Road To Cree, The	DCDEFFFEFFBFAFECDCDEFFFEAFECCBBCDCDEFFFEFABAAFECD...	0.8	1	0.453472222222222	5361
8	17297	Gin I Were Where Gadie Rins	AABDDEFFFEDFFFEDEEFEDBADABDADEFFFEFAAABCDFDB	0.8	1	0.413636363636364	6411
9	18349	Let Down The Blade	FGAABAABAAFDFAAAFDDEFGGGFGAABAABAAFDGGGBCDBCAF...	0.8	1	0.246938775510204	8536
10	10337	McKenna"s #2	FAAFGFEDBCDEFGAFGEBBBGEDBBABCDCDEFAAFGFEDBCDEFG...	0.8	1	0	8709
11	18335	Noel Hill"s	FFFEFDDEFAAFEEDEFFFEFDDEFAAFEEEE	0.8	1	0	8709
12	19534	Rabbit, The	DDBBAGABDEEEEDEGGABAGAGFGEDEFGEDBGABC	0.8	1	0	8709
13	16773	Turtledove, The	CDEEEDCCCDEDEFEEEFGGGGAAAAEEEEEECD	0.7	13	0.781176472762052	4

Query executed successfully. | PODGE-PC7\DEVELOPER (10.0 SP1) | Podge-PC7\Podge (54) | Test | 00:00:01 | 11944 rows

Figure 37: Results of the SQL query containing Semex and Jaro-Winkler scores with ranks ordered by Semex rank

Figure 37 shows the results of the SQL query where the string 'CDEEEDEGGA' is compared to the whole corpus of tunes returning Jaro-Winkler and Semex scores with these scores ranked. These results are sorted by Semex rank in ascending order. In this case the top 12 results all score 0.8 and are ranked joint 1st. The tune "The Turtledove" is given a rank of 13 as it has the next highest Semex score. Note how this tune has a Jaro-Winkler rank of 4, a much higher Jaro-Winkler rank than any of the tunes above it, most of which have a Jaro-Winkler rank in the thousands.

By contrast, Figure 38 shows the results of the same SQL query ordered by Jaro-Winkler rank instead of Semex rank. The top ranked Jaro-Winkler result is given a rank of 2475 by the Semex algorithm. Although some deviation between algorithms was expected, it was not expected at this level.

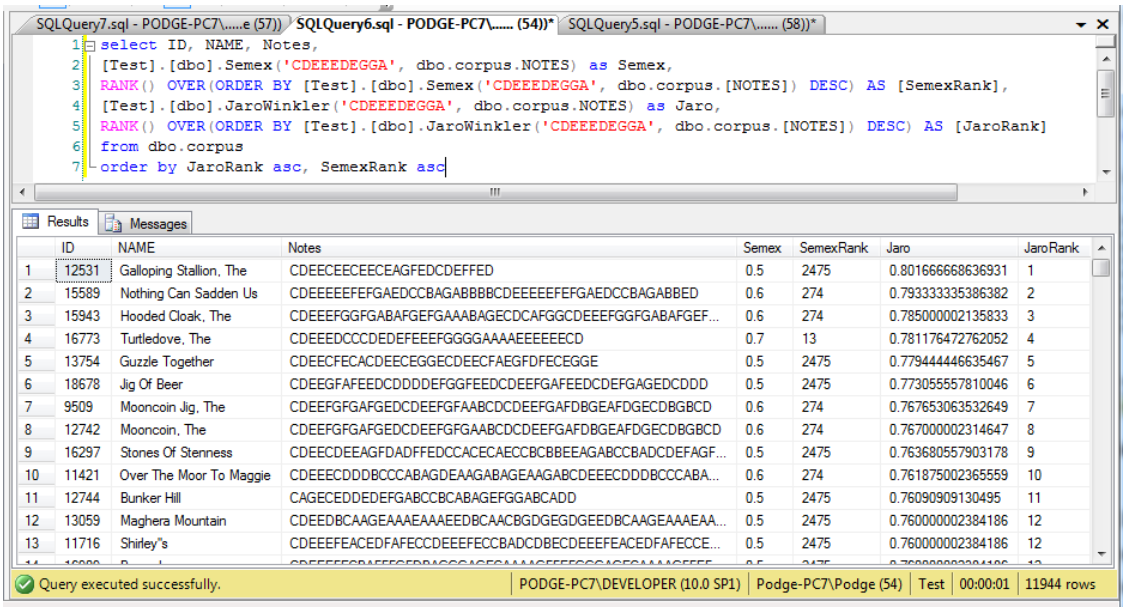


Figure 38: Results of the SQL query containing Semex and Jaro-Winkler scores with ranks ordered by Jaro-Winkler rank

The result of this experiment is twofold;

- It is possible to formulate SQL queries based on string distance functions and rank the results accordingly.
- String distance algorithms may agree or disagree on the result of a comparison.

It became clear that an experiment in combining ranks from different string distance functions would also need to be conducted. In order to do this, two further custom functions were created, normalisedRank and stdevMusic. Code for both of these functions is available in Appendix D. The formula to normalise the ranks is as follows;

$$1-\left(\frac{sr-4}{mr*4}\right)$$

Where sr is the sum of all four ranks and mr is the maximum rank possible. The normalisedRank function takes 4 ranks and the maximum rank possible (the total number of records in the corpus, 11944) as arguments. This function then combines the ranks using the following C# code;

```
        public double GetNormalisedRank(int firstValue, int secondValue, int
thirdValue, int fourthValue, int count)
        {
            // need four ranks and the total count of records (highest rank) to
normalise
            int sum = firstValue + secondValue + thirdValue + fourthValue;
            double normalisedRank = 1.0 - ((sum - 4.0) / (count * 4.0));
```

```
            return normalisedRank;
        }
```

Different combinations of ranks could return the same combined rank score. The figure below shows that Tune 1 and Tune 2 received different ranks from different algorithms but were given the same combined normalised score.

	Rank 1	Rank 2	Rank 3	Rank 4	Sum	stdev	Total Score Normalised	Average
Tune 1	1	1	1	10	13	4.5	0.999811621	3.25
Tune 2	3	3	3	4	13	0.5	0.999811621	3.25
Tune 3	10000	20	10000	1	20021	5767.44573	0.581023945	5005.25
Total Tunes	11944							

Figure 39: Combined rank score calculation

The stdev column in the figure above shows that the ranks for Tune 1 deviate more than for Tune 2. Tune 2 therefore represents a better match if standard deviation is considered.

The following table shows the C# code used to calculate the standard deviation of the ranks;

Table 40: Commonly available C# code used to calculate standard deviation

```
        /// <summary>
        /// gets the stdev of the four values passed to it.
        /// </summary>
        /// <param name="firstValue"></param>
        /// <param name="secondValue"></param>
        /// <param name="thirdValue"></param>
        /// <param name="fourthValue"></param>
        /// <returns>a value between 0-1 of the similarity</returns>
        public override double GetSimilarity(double firstValue, double
secondValue, double thirdValue, double fourthValue)
        {

            ArrayList rankList = new ArrayList();
            rankList.Add(firstValue);
            rankList.Add(secondValue);
            rankList.Add(thirdValue);
            rankList.Add(fourthValue);

            return StandardDeviation(rankList);

        }
        ///<Summary>
        ///Calculates standard deviation of numbers in an ArrayList
        ///</Summary>
        public static double StandardDeviation(ArrayList num)
        {
            double SumOfSqrs = 0;
```

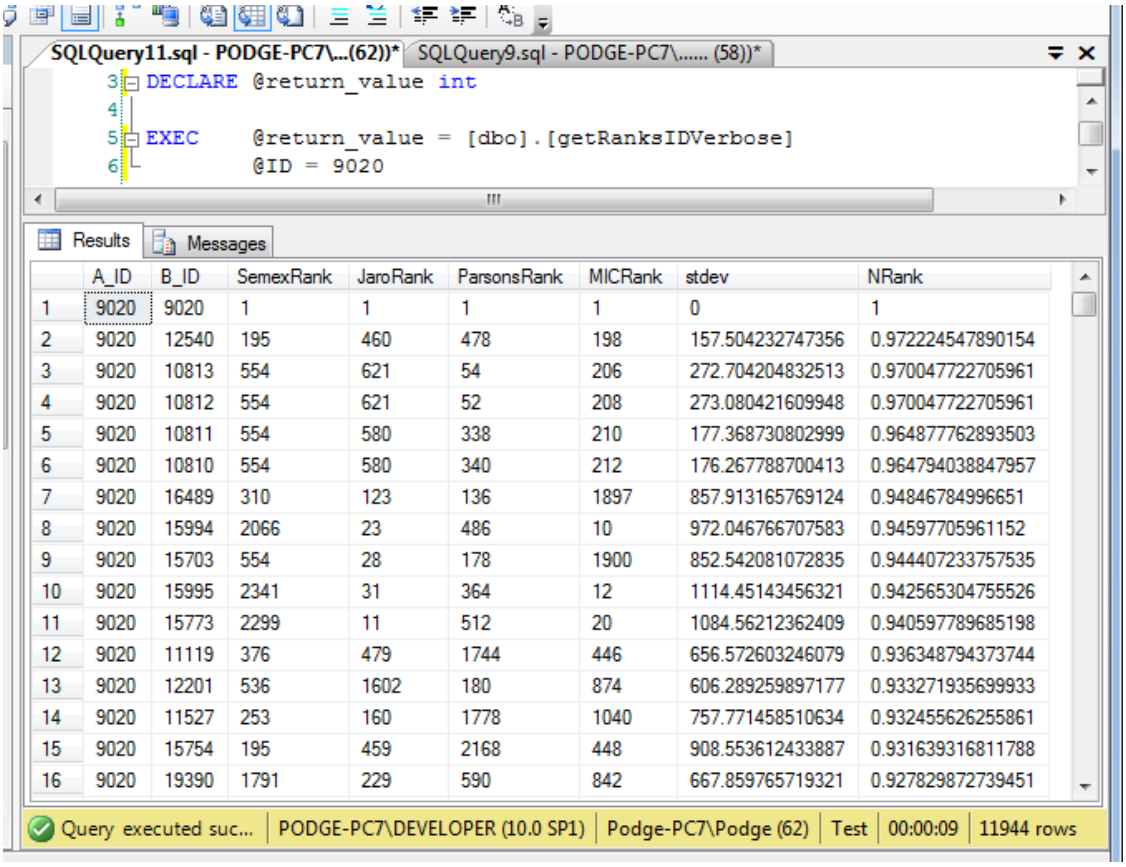
```
            double avg = Average(num);
            for (int i = 0; i < num.Count; i++)
            {
                SumOfSqrs += Math.Pow(((double)num[i] - avg), 2);
            }
            double n = (double)num.Count;
            return Math.Sqrt(SumOfSqrs / (n - 1));
        }

        ///<Summary>
        ///Calculates average of numbers of integer data type in an ArrayList
        ///</Summary>
        public static double Average(ArrayList num)
        {
            double sum = 0.0;
            for (int i = 0; i < num.Count; i++)
            {
                sum += (double)num[i];
            }
            double avg = sum / System.Convert.ToDouble(num.Count);

            return avg;
        }
```

Two stored procedures were then created in order to carry out combined ranking experiments. The first was called getRanksIDVerbose and the second called getRanksID. The verbose version returns the individual string distance algorithm ranks, the combined rank and the standard deviation, the second, getRanksID performs exactly the same calculations as the first but only returns the combined rank and the standard deviation scores.

Figure 40 shows the results of comparisons between the tune with ID 9020 and the rest of the corpus. As one would expect, tune 9020 is a perfect match with itself and receives four rankings of 1st. This results in a combined normalised rank of 1 (the NRank column on the right) and a standard deviation of 0. Tune 12540 receives second place with an NRank score of 0.972. The tune parts with ID's 10813 and 10812 (rows 3 and 4) receive exactly the same NRank score, however, because tune ID 10813 (row 3) has a lower standard deviation it is placed higher than tune ID 10812 (row 4).



	A_ID	B_ID	SemexRank	JaroRank	ParsonsRank	MICRank	stdev	NRank
1	9020	9020	1	1	1	1	0	1
2	9020	12540	195	460	478	198	157.504232747356	0.972224547890154
3	9020	10813	554	621	54	206	272.704204832513	0.970047722705961
4	9020	10812	554	621	52	208	273.080421609948	0.970047722705961
5	9020	10811	554	580	338	210	177.368730802999	0.964877762893503
6	9020	10810	554	580	340	212	176.267788700413	0.964794038847957
7	9020	16489	310	123	136	1897	857.913165769124	0.94846784996651
8	9020	15994	2066	23	486	10	972.046766707583	0.94597705961152
9	9020	15703	554	28	178	1900	852.542081072835	0.944407233757535
10	9020	15995	2341	31	364	12	1114.45143456321	0.942565304755526
11	9020	15773	2299	11	512	20	1084.56212362409	0.940597789685198
12	9020	11119	376	479	1744	446	656.572603246079	0.936348794373744
13	9020	12201	536	1602	180	874	606.289259897177	0.933271935699933
14	9020	11527	253	160	1778	1040	757.771458510634	0.932455626255861
15	9020	15754	195	459	2168	448	908.553612433887	0.931639316811788
16	9020	19390	1791	229	590	842	667.859765719321	0.927829872739451

Figure 40: Combined ranks with standard deviation

Testing the results of this experiment by means of an online survey was conducted in the next phase of this experiment.

6.4.4 Phase 3 – Testing the combined ranking system on humans

As a result of feedback from participants of the previous survey, the number of tune pairs was reduced for the second online survey. Six tune pairs were chosen at random instead of ten reducing the amount of time taken to complete the survey to about five minutes. No distinction was made between experts and non-experts in Irish music for the second survey as they tended to vote similarly in the first survey. The survey was available online at the following web address for the participants to complete - http://fluidsurveys.com/surveys/podge/irish-music-similarities-2-1/ (Lavin 2010)

Six pairs of tunes were chosen as follows;

- Two pairs with a high combined ranking and low standard deviation score (reliable similarity)

- Two Pairs with a high combined ranking and high standard deviation score (unreliable similarity)
- One pair with a low combined ranking and low standard deviation score (reliable dissimilarity)
- One pair with a low combined ranking and a high standard deviation score (unreliable dissimilarity)

Figure 41 below shows how tune pairs were chosen for the second online survey.

	A	B	C	D	E	F	G	H	I	J
1	**Pair No**	**Tune**	**Name**	**Part**	**Measure**	**Key**	**stdev**	**Nrank**	**Similar**	**Reliable**
2	1a	8921	Hag at the Churn	1	6/8	Dmix				
3	1b	9213	Buck in the wood	3	6/8	Gmaj				
4							1929.559	0.917050402	Yes	No
5										
6	2a	14884	Fred Finns	1	4/4	Dmaj				
7	2b	12590	McDonagh's	2	4/4	Edor				
8							267.4700918	0.610913429	No	Yes
9										
10	3a	8969	The Mossy Banks	4	4/4	Gmaj				
11	3b	11437	The Traveller	2	4/4	GMaj				
12							219.1863743	0.98986939	Yes	Yes
13										
14	4a	11248	Humours of Whiskey	1	9/8	Bmin				
15	4b	12877	The Banks of Allan	1	6/8	DMaj				
16							4233.944605	0.657003516	No	No
17										
18	5a	14018	Lad O'Beirnes	1	4/4	GMaj				
19	5b	15871	Greigs Pipes	3	4/4	GMaj				
20							18.62793601	0.998869725	Yes	Yes
21										
22	6a	10764	Colonel Frazer	6	2/2	GMaj				
23	6b	10612	The West Wind	4	2/2	GMaj				
24							1851	0.922429672	Yes	No

Figure 41: Survey 2 tune pairs with ranking and stdev scores

Twenty participants responded to the survey and responded as shown in Figure 42. Weighting of votes was not carried out for the initial count. Where a participant voted that a pair was “similar” or “very similar” that vote was counted as simply “similar” and where a participant voted that a pair was “different” or “very different” that vote was counted as “dissimilar”.

A	B	C	D	E	F	G
Name	**Question 1**	**Question 2**	**Question 3**	**Question 4**	**Question 5**	**Question 6**
Brian Joseph Finian Duggan	Similar	Different	Very similar	Different	Similar	Very similar
Pádhraic Ó Súilleabháin	Different	Different	Very similar	Different	Similar	Very similar
Frances McMahon	Very different	Similar	Very similar	Very different	Very similar	Very similar
terry lavin	Different	Different	Similar	Very different	Similar	Different
paul browne	Very different	Very different	Similar	Very different	Similar	Very similar
Deirdre Smyth	Very different	Different	Very similar	Different	Similar	Very similar
Alan O'Brien	Different	Very similar	Very similar	Very different	Very similar	Very similar
Diarmuid Cooke	Similar	Different	Very similar	Very different	Very similar	Very similar
Jim Mulhall	Different	Similar	Very similar	Similar	Very similar	Very similar
Treasa Lavin	Very different	Very different	Very similar	Very different	Similar	Very similar
john breen	Different	Similar	Very similar	Very different	Similar	Very similar
Paul Boylan	Different	Different	Very similar	Similar	Very similar	Very similar
Martin Denning	Different	Different	Very similar	Very different	Similar	Very similar
irene dunne	Different	Similar	Very similar	Very different	Similar	Very similar
Aisling Ní Ghrádaigh	Different	Different	Very similar	Similar	Very similar	Similar
Eibhlís O'Sullivan	Similar	Very similar	Very similar	Different	Similar	Very similar
Ruth Maher	Very different	Very similar	Very similar	Different	Very similar	Very similar
Caroline Bermingham	Very different	Similar	Very similar	Very different	Very similar	Very similar
oisin o grady	Very different	Similar	Very similar	Very different	Similar	Very similar
aimee o reilly	Different	Similar	Very similar	Very different	Similar	Very similar
Similar count	3	10	20	3	20	19
Dissimilar count	17	10	0	17	0	1
Result	**Dissimilar**	**Unknown**	**Similar**	**Dissimilar**	**Similar**	**Similar**

Figure 42: Online Survey 2 Responses

Voting produced in the following result;

Table 41: Results of Online Survey 2

	Humans	**Computer**
Pair 1	Dissimilar	Unreliable similarity
Pair 2	Unknown	Reliable dissimilarity
Pair 3	Similar	Reliable similarity
Pair 4	Dissimilar	Unreliable dissimilarity
Pair 5	Similar	Reliable similarity
Pair 6	Similar	Unreliable similarity

In order to discern if the human participants voted that pair 2 were similar or dissimilar weighting of votes was carried out.

Table 42: Vote weighting scores

Very different	-2
Different	-1
I don't know	0

Similar	1
Very similar	2

A tune pair receiving a score below zero means that participants have voted that the tunes are different and above zero meaning that the tunes are similar.

Name	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6
Brian Joseph Finian Duggan	1	-1	2	-1	1	2
Pádhraic Ó Súilleabháin	-1	-1	2	-1	1	2
Frances McMahon	-2	1	2	-2	2	2
terry lavin	-1	-1	1	-2	1	-1
paul browne	-2	-2	1	-2	1	2
Deirdre Smyth	-2	-1	2	-1	1	2
Alan O'Brien	-1	2	2	-2	2	2
Diarmuid Cooke	1	-1	2	-2	2	2
Jim Mulhall	-1	1	2	1	2	2
Treasa Lavin	-2	-2	2	-2	1	2
john breen	-1	1	2	-2	1	2
Paul Boylan	-1	-1	2	1	2	2
Martin Denning	-1	-1	2	-2	1	2
irene dunne	-1	1	2	-2	1	2
Aisling Ní Ghrádaigh	-1	-1	2	1	2	1
Eibhlís O'Sullivan	1	2	2	-1	1	2
Ruth Maher	-2	2	2	-1	2	2
Caroline Bermingham	-2	1	2	-2	2	2
oisin o grady	-2	1	2	-2	1	2
aimee o reilly	-1	1	2	-2	1	2
Totals	**-21**	**1**	**38**	**-26**	**28**	**36**
Result	**Dissimilar**	**Similar**	**Similar**	**Dissimilar**	**Similar**	**Similar**

Figure 43: Weighted scores for Survey 2

6.4.4.1 Analysis of results

Figure 43 shows how the weighting of scores results in pair 2 being voted similar by the finest of margins. Because participants voted that pair 2 was similar by just one point the result is too close to be relied upon. The final results are shown in Table 43.

Table 43: Online Survey 2 Final Result

	Humans	**Computer**
Pair 1	Dissimilar	Unreliable similarity
Pair 2	Similar (unreliable)	Reliable dissimilarity
Pair 3	Similar	Reliable similarity

Pair 4	Dissimilar	Unreliable dissimilarity
Pair 5	Similar	Reliable similarity
Pair 6	Similar	Unreliable similarity

The computer algorithm and human participants disagreed about the result of pair 1 significantly. One reason for this is that three of the four computer algorithms make transposition invariant comparisons. The tunes in this pair were in the keys of D mixolydian and G major and this may have prevented the participants from recognising similarities between the tunes. In order to ascertain whether this had an effect on the result a further survey may be necessary with the tunes in the keys of G mixolydian (converted from D mixolydian) and G major respectively.

Pair 2 consisted of tunes in different keys, had an average combined ranking score of 0.61 and a low standard deviation. The initial vote was tied and after the scores were weighted the final result was that the tunes were similar by a margin of just 1 point. This result is unreliable. The computer algorithm result suggests that because the standard deviation is low the normalised combined ranked score of 0.61 should be reliable. It would seem that humans are undecided on the similarity of two tunes when their combined ranking score is below a certain threshold and that a score of 0.61 is within this range. The result suggests that a score of 0.61 could represent a tune pair that is similar or dissimilar. Mapping this threshold has been identified as an area for further research, investigation and future work.

The computer algorithm agreed with the humans for pairs 3, 4, 5 and 6. This represents a significant improvement on the first online survey. The computer algorithm suggested that the comparison for pair 1 was unreliable. The algorithm suggested that pair 2 were dissimilar however the survey participants were undecided, voting the pair to be similar by a margin of just 1 point. Disregarding pair 2 the algorithm agreed with humans 80% of the time (4 out of 5 pairs) compared with 60% of the time in Section 6.3.8.

6.4.5 Phase 4 – Constructing Similarity Matrices

The experiments carried out in this final phase of the project were based on the results of previous experiments and all of the previous research carried out in earlier phases of the project.

6.4.5.1 Parsons Code and Breathnach MIC Similarity Matrices

A method for constructing matrices based on Breathnach's Melodic Indexing code and Parsons Code were introduced in Section 6.2.7 and Section 6.4.2 using both the Java platform and the Microsoft C# platform. A portion of matrices for both Parsons Code and Melodic Indexing Code can be seen in Figure 36.

Dynamic Parsons Code and Melodic Indexing Code similarity matrices of the entire corpus may be constructed in a few seconds using the SQL code in Table 37.

6.4.5.2 Jaro-Winkler Similarity Matrix

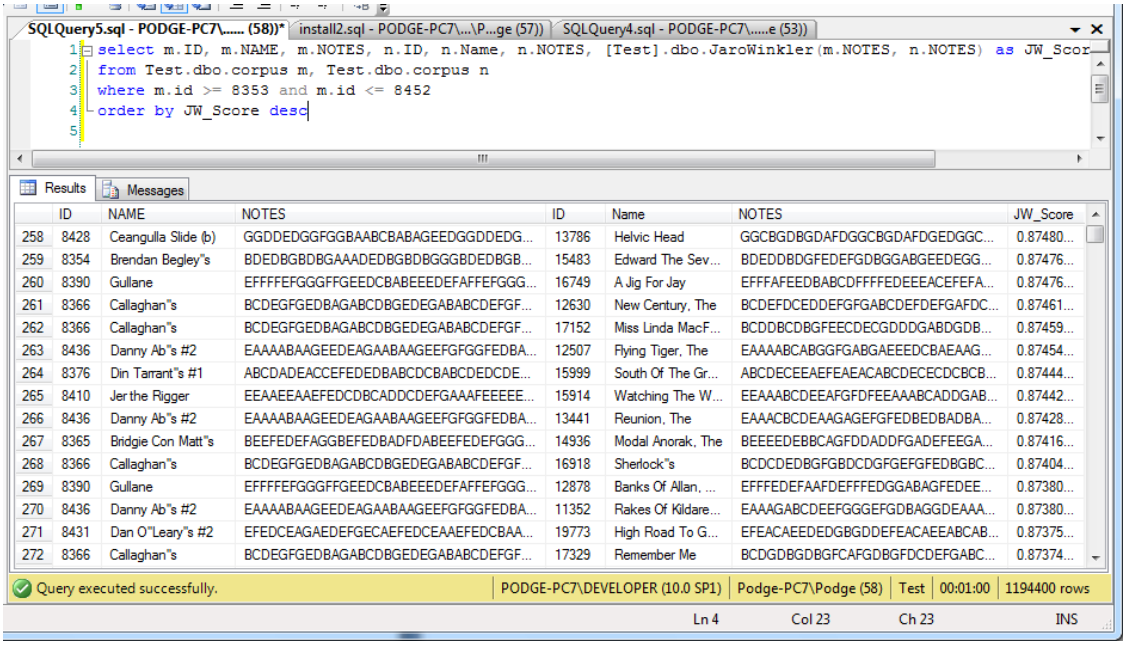
This first experiment attempted to construct a similarity matrix using just one SQL query. It involved the use of a SQL query that joins a table to itself in order to iterate through all records in the corpus database table using the Jaro-Winkler function to compare each individual row with all others.

In order to estimate the time taken to execute the SQL query a subset of 100 records were compared against all others in the corpus. The SQL query in Table 44 was used for this purpose.

Table 44: SQL to compare 100 tunes to a corpus using Jaro-Winkler

```
select m.ID, m.NAME, m.NOTES, n.ID, n.Name, n.NOTES,
[Test].dbo.JaroWinkler(m.NOTES, n.NOTES) as JW_Score
from Test.dbo.corpus m, Test.dbo.corpus n
where m.id >= 8353 and m.id <= 8452
order by JW_Score desc
```

Figure 44 shows that it took exactly 1 minute to compare 100 tunes with 11944 other tunes, a total of 1194400 comparisons.



	ID	NAME	NOTES	ID	Name	NOTES	JW_Score
258	8428	Ceangulla Slide (b)	GGDDEDGGFGGBAABCBABAGEEDGGDDEDG...	13786	Helvic Head	GGCBGDBGDAFDGGCBGDAFDGEDGGC...	0.87480...
259	8354	Brendan Begley"s	BDEDBGBDBGAAADEDBGBDBGGGBDEDBGB...	15483	Edward The Sev...	BDEDDBDGFEDEFGDBGGABGEEDEGG...	0.87476...
260	8390	Gullane	EFFFFEFGGGFFGEEDCBABEEEDEFAFFEFGGG...	16749	A Jig For Jay	EFFFAFEEDBABCDFFFFEDEEEACEFEFA...	0.87476...
261	8366	Callaghan"s	BCDEGFGEDBAGABCDBGEDEGABABCDEFGF...	12630	New Century, The	BCDEFDCEDDEFGFGABCDEFDEFGAFDC...	0.87461...
262	8366	Callaghan"s	BCDEGFGEDBAGABCDBGEDEGABABCDEFGF...	17152	Miss Linda MacF...	BCDDBCDBGFEECDECGDDDGABDGDB...	0.87459...
263	8436	Danny Ab"s #2	EAAAABAAGEEDEAGAABAAGEEFGFGGFEDBA...	12507	Flying Tiger, The	EAAAABCABGGFGABGAEEEDCBAEAAG...	0.87454...
264	8376	Din Tarrant"s #1	ABCDADEACCEFEDEDBABCDCBABCDEDCDE...	15999	South Of The Gr...	ABCDECEEAEFEAEACABCDECECDCBCB...	0.87444...
265	8410	Jer the Rigger	EEAAEEAAEFEDCDBCADDCDEFGAAAFEEEEE...	15914	Watching The W...	EEAAABCDEEAFGFDFEEAAABCADDGAB...	0.87442...
266	8436	Danny Ab"s #2	EAAAABAAGEEDEAGAABAAGEEFGFGGFEDBA...	13441	Reunion, The	EAAACBCDEAAGAGEFGFEDBEDBADBA...	0.87428...
267	8365	Bridgie Con Matt"s	BEEFEDEFAGGBEFEDBADFDABEEFEDEFGGG...	14936	Modal Anorak, The	BEEEEDEBBCAGFDDADDFGADEFEEGA...	0.87416...
268	8366	Callaghan"s	BCDEGFGEDBAGABCDBGEDEGABABCDEFGF...	16918	Sherlock"s	BCDCDEDBGFGBDCDGFGEFGFEDBGBC...	0.87404...
269	8390	Gullane	EFFFFEFGGGFFGEEDCBABEEEDEFAFFEFGGG...	12878	Banks Of Allan, ...	EFFFEDEFAAFDEFFFEDGGABAGFEDEE...	0.87380...
270	8436	Danny Ab"s #2	EAAAABAAGEEDEAGAABAAGEEFGFGGFEDBA...	11352	Rakes Of Kildare...	EAAAGABCDEEFGGGEFGDBAGGDEAAA...	0.87380...
271	8431	Dan O"Leary"s #2	EFEDCEAGAEDEFGECAEFEDCEAAEFEDCBAA...	19773	High Road To G...	EFEACAEEDEDGBGDDEFEACAEEABCAB...	0.87375...
272	8366	Callaghan"s	BCDEGFGEDBAGABCDBGEDEGABABCDEFGF...	17329	Remember Me	BCDGDBGDBGFCAFGDBGFDCDEFGABC...	0.87374...

Figure 44: Result of the SQL query comparing 100 tunes to the corpus

An estimated 142 million comparisons would need to be performed in order to construct the entire matrix resulting in an estimated execution time of 120 minutes.

The SQL query below was used to perform comparisons on the whole corpus and instead of returning them to the Management Studio Console, they were stored in a database table named JaroWinklerMatrix.

Table 45: SQL query for constructing the Jaro-Winkler Matrix

```
select m.ID, m.NAME, m.NOTES, [Test].dbo.JaroWinkler(m.NOTES,
n.NOTES) as JW_Score
into [Test].dbo.JaroWinklerMatrix
from Test.dbo.corpus m, Test.dbo.corpus n
order by JW_Score desc
```

The SQL query above completed the similarity matrix in less than 45 minutes. The screenshot below shows that 142,659,136 rows were inserted into the database table JaroWinklerMatrix. This represents the amount of records in the corpus squared i.e. 11944^2. This table requires 4.5GB of hard disk storage.

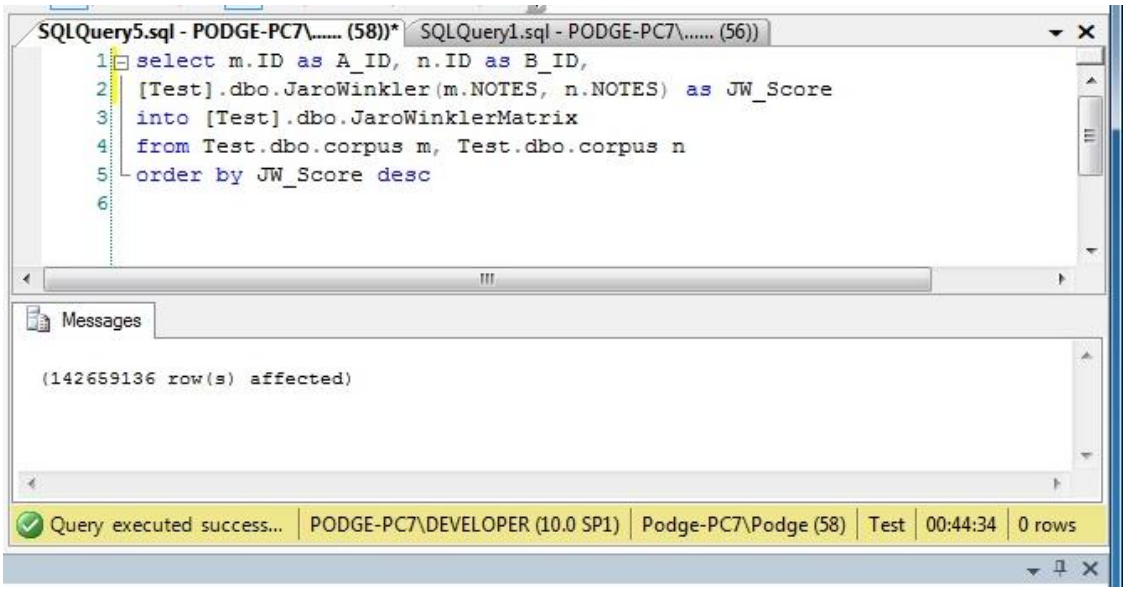


Figure 45: Completed Jaro-Winkler Similarity Matrix

The completion of this similarity matrix represents the delivery of the secondary objective for this project – the construction of a similarity matrix.

Using this method, similarity matrices based on the Levenshtein and Semex algorithms are also possible by substituting the appropriate function name at Line 2 of Figure 45 e.g. `[Test].dbo.Levenshtein(m.NOTES, n.NOTES) as Levenshtein_Score` for a Levenshtein similarity matrix or `[Test].dbo.Semex(m.NOTES, n.NOTES) as Semex_Score` for a Semex based similarity matrix.

This method illustrates the power of SQL to dynamically create similarity matrices using a variety of algorithms on a corpus of tunes of unknown size and content.

6.4.5.3 Similarity matrix using the Combined Ranking System

Following the success of the JaroWinkler similarity matrix the next experiment attempted to create a similarity matrix using the combined ranking method developed in Section 4.3. Two stored procedures were developed, getRanksID and CalculateMatrix, in order to iterate through all of the tune parts in the corpus, comparing each of them to all of the tune parts in the corpus and store the results in a database table. Both of these stored procedures are available in Table 60 and Table 61 of Appendix D.

The getRanksID stored procedure compares a tune part to itself and all other tune parts in the corpus. When passed a tune part ID as an argument it returns 11944 rows, each containing the normalised combined rank score with the standard deviation between the four algorithms.

The calculateMatrix stored procedure iterates through all tune ID's in the corpus, sends the ID to the getRanksID stored procedure and stores the results in a database table. MS SQL 2008 allows for the insertion of multiple rows of data returned from a stored procedure being stored in a database table using just one insert statement. The transact SQL code in Table 47 inserts all of the rows returned by the getRanksID stored procedure without having to iterate through all 11944.

Table 46: Database cursor that iterates through all tune parts by ID

```
DECLARE tune_cursor CURSOR FOR SELECT cast([ID] as int) as ID, NOTES
FROM [Test].[dbo].[corpus] where (ID >= 8353 and ID <= 20297) order
by ID asc
```

Table 47: T-SQL INSERT code to store comparison results.

```
INSERT dbo.matrix (A_ID, B_ID, STDEV, NRank)
EXEC  @return_value = [dbo].[getRanksID]
@ID = @corpusID
```

This method of constructing a similarity matrix is not as elegant as the method used to construct the Jaro-Winkler matrix. In order for this to be possible, the getRanksID stored procedure must take two tune ID's as arguments. An investigation into adapting the getRanksID stored procedure in this manner revealed that it would result in serious performance problems. In order to calculate score, rank and standard deviation in the Combined Ranking System of assessing similarity, score rank and standard deviation for the whole corpus must first be calculated. It does not make sense to return only 1 row from a getRanksID stored procedure taking two arguments of tune ID's for comparison and discarding all other 11943 scores.

A trial run of the calculateMatrix stored procedure revealed that the laptop running the experiments had insufficient memory to complete the task in one go so the task was divided into stages. The combined ranking matrix was completed by iterating through groups of 2000 tune parts at a time. This was done over the course of a few days. The

matrix was completed without issue and the resulting database table is about 8.5GB in size.

6.5 Conclusion

This chapter described how string distance experiments on ABC notation data were designed and carried out. An explanation of how raw data was imported, cleaned and stored in a relational database was also offered. A brief description of the Java and C Sharp programming frameworks used to carry out experiments was given. Details of Levenshtein and Jaro-Winkler comparison and distribution experiments are described. This chapter also outlined an attempt to construct a similarity matrix using the Semex algorithm and how it was halted due to performance problems. A successful experiment carried out on the Java platform in order to construct a computerised version of Breathnach's Melodic Indexing System is illustrated.

This chapter continued by outlining various experiments carried out on the Microsoft dotNet platform. These experiments included the testing of existing and new custom string distance functions in the SimMetrics C Sharp library and the testing of a Combined Ranking System. A description of how participants were surveyed is presented before concluding the chapter with a description of how four similarity matrices were proposed and constructed.

7. CONCLUSION

7 Introduction

This chapter summarises the research domain and describes the research carried out throughout this project. Descriptions of how contributions were made to the body of knowledge are presented. The experimentation and evaluation phases are discussed followed by an examination of the scope of the project limitations. Research objectives that were achieved are outlined. Areas for further investigation, future work and research areas are identified. Some final conclusions complete this chapter.

7.1 Research Definition & Research Overview

The research for this project focused on the evaluation and improvement of string distance algorithms in order to identify similarities in the corpus of Irish traditional music. A secondary aim was to design a process by which an Irish music similarity matrix could be constructed.

Numerous string distance algorithms were evaluated for suitability purposes before deciding on candidates. Two alternative methods of assessing similarity invented in the 1960's and 1970's, Breathnach's Melodic Indexing Code and Parsons Code, were studied, computerised and converted into computer algorithms.

Research into how results from both types of algorithms could be combined was undertaken. A Combined Ranking System (CRS) was then developed and tested on survey participants.

7.2 Contributions to the Body of Knowledge

Five contributions to the body of knowledge were made over the course of this project.

7.2.1 Contribution 1 - Weighting Melodic Sequence Variation

Irish musicians commonly vary the manner in which melodies are played. This can lead to string distance algorithms penalising phrases of notes because they contain

notes played in an alternative but correct sequence. This contribution allows compensation scores for these alternative note sequences.

7.2.2 Contribution 2 - Weighting Tune Prefixes

Some traditional Irish tunes are played after short introductory prefixes consisting of two or more notes. This contribution allows for the recognition of these initial notes by implementing increased scoring for matching opening notes.

7.2.3 Contribution 3 – Computerising Breathnach's & Parsons' Systems

Breandán Breathnach and Denys Parsons introduced two different systems for assessing similarity in the 1960's and 1970's respectively, the Melodic Indexing System and Parsons Code. Both of these systems were examined and computerised for the purposes of inclusion in a Combined Ranking System used to compare music and to construct a similarity matrix for Irish traditional music.

7.2.4 Contribution 4 – Improvements to the Melodic Indexing System

The following improvements to Breandán Breathnach's Melodic indexing system were proposed;

- Sorting index codes alphabetically instead of numerically thus allowing the comparison of different length codes.
- A system of using distance and normalisation was designed and introduced. This allows the return of a normalised MIC score similar to scores returned by string distance algorithms.

7.2.5 Contribution 5 – A Combined Ranking System

All of the string distance algorithms used to make comparisons between strings of musical notes returned a normalised measure of similarity between 0 and 1. Two further algorithms were developed based on Breathnach's MIC and Parsons Code that also returned normalised similarity scores between 0 and 1. This enabled the ranking of scores returned by all types of algorithm. A system was then developed that combined the ranks returned by all algorithms. The standard deviation between ranks was also returned.

7.3 Experimentation, Evaluation and Limitation

7.3.1 Experimentation

Various string distance experiments were carried out on a corpus of Irish traditional dance music tune parts in ABC notation. These experiments incorporated all five contributions described earlier. The results of these experiments were analysed and used to define a process by which a similarity matrix could be constructed.

7.3.2 Evaluation

During the evaluation stage, humans were surveyed twice in order to ascertain if they agreed with the results of computer algorithms. The results of the first survey were analysed and evaluated. Proposals for improvements to the string distance algorithms were formulated and implemented. Some string distance algorithms were also improved by considering music theory and then tested on humans by means of a second online survey.

In the second online survey the following hypothesis was tested: if multiple different algorithms rank a comparison similarly, can that comparison be assumed as accurate? The conclusion drawn from the results of the experiment is that yes, if multiple different algorithms rank a result similarly then that result is more accurate than using string distance algorithms individually.

7.3.3 Limitations

Similarity comparison experiments were performed on music data that contained melody, time signature, musical key, title but no playing style data. Similarity was assessed primarily on melody.

Approximately half of the source data was deemed unreliable as it did not comply with the ABC notation specification. This data was discarded as considerable manual resources would be needed in order to correct the erroneous ABC files.

Similarity matrices were constructed by recording different types of comparisons between tune parts in a database. These databases are currently limited to being

queried by using SQL queries and this requires specialist knowledge. A better means of querying these databases has been identified as an area for further development and future work.

7.4 Future Work & Research

A number of areas have been identified for further investigation, future work and research. These include;

7.4.1 Parsons Code & Melodic Index Code Precision

In Section 2.3.1 two terms were defined to describe two methods of calculating distance, MICRank and MICDenseRank. An opportunity to increase the accuracy of MIC and Parsons Code scores was also identified. This task involves calculating individual distances from a match (MICDenseRank) instead of the current method of assigning the same distance from a match to a pair either side of the match (MICRank).

Calculating individual distances from a match is a departure from the original system and will need to be programmed, tested and evaluated as part of future research.

7.4.2 Jaro-Winkler matching prefixes

A feature of the Jaro-Winkler algorithm was identified that could have a possible application in the Irish music domain. This feature was utilised when comparing sequences of musical notes, however, its positive or negative effectiveness was not measured. In order to take advantage of the concept of matching prefixes further investigation, examination and testing is necessary.

7.4.3 Similarity / Dissimilarity threshold

While analysing the results of the second online survey it became apparent that humans were undecided if a particular tune pair were similar or dissimilar. The score returned by the Combined Ranking System for this pair was near the centre of the distribution making it unclear if the computer algorithm was indicating similarity or dissimilarity. When scores are returned at either end of the spectrum, between 0 and 0.2 and between 0.8 and 1, dissimilarity and similarity respectively may easily be

inferred. The closer the score is to the centre of the distribution, the more difficult it is to predict whether humans feel that a tune pair is similar or different. The need to establish the threshold scores where humans felt that tunes were similar or dissimilar was identified as an area that warrants further investigation.

7.4.4 User querying and surveying

The similarity matrices built using the methods and processes defined during this project cannot be easily queried by persons that are not skilled in SQL. The necessity to develop a desktop application, website or mobile application that allows users to easily query matrices and record feedback has been identified as essential future work and development.

7.5 Conclusion

7.5.1 Objectives

The following project objectives were achieved;

- The identification of suitable string distance algorithms for the purposes of comparing music in ABC notation.
- To improve specific string distance algorithms by implementing features unique to music theory.
- To survey humans in order to assess if their choices agreed with computer algorithms.
- Multiple similarity matrices were constructed.

7.5.2 Deliverables

The following deliverables were accomplished;

- A process was designed for comparing Irish traditional dance tunes. This Combined Ranking System was built on improvements to string distance algorithms and tested on humans.
- Similarity matrices were constructed using four different methods of comparison.

7.5.3 Conclusion

This project has strived to solve the problem of identifying similarities in Irish music by investigating, evaluating and improving different methods of assessing musical likeness. A system was produced in line with the project objective and aims that allowed for a similarity matrix for Irish traditional dance music to be constructed.

Music Information Retrieval (MIR) and string distance comparison remain lively research topics. This project has identified multiple areas that require future work and further study. Two areas are of primary importance to the author, the collection of musical similarity data by means of a mobile or social networking application for the purposes of surveying humans and associated research and enabling the navigation and querying of similarity matrices by means of a website, mobile or desktop application.

"Where words leave off, music begins"
Heinrich Heine, 1797-1856

BIBLIOGRAPHY

Allan, H. & Wiggins, G., 2006. Further aspects of similarity. In *Proceedings of the 2nd Digital Music Research Network Summer Conference*.

Black, B., 2010. Bill Blacks Web ABC Homepage. *Bill Blacks web ABC tune collections*. Available at: http://www.qmcorp.net/webabc/collections/index.html [Accessed April 20, 2010].

Breathnach, B., 1963. *Ceol rince na hÉireann Cuid I [Dance Music of Ireland] Vol I*, Oifig an tSolathair.

Breathnach, B., 1976. *Ceol rince na hÉireann Cuid II [Dance Music of Ireland] Vol II*, Oifig an tSoláthair.

Breathnach, B., 1985. *Ceol rince na hÉireann Cuid III [Dance Music of Ireland] Vol III*, An Gum.

Breathnach, B., 1996. *Ceol rince na hÉireann Cuid IV [Dance Music of Ireland] Vol IV*, An Gum.

Breathnach, B., 1999. *Ceol rince na hÉireann Cuid V [Dance Music of Ireland] Vol V*, An Gum.

Breathnach, B., 1982. Between the Jigs and the Reels. *Ceol V*, 2. Available at: http://msikio.online.fr/Breathnach/breandn.htm [Accessed April 23, 2010].

Budzinsky, C., 1991. Automated spelling correction. *Statistics Canada*.

Bunting, E., 1969. *The Ancient Music of Ireland. An edition comprising the three collections by Edward Bunting originally published in 1796, 1809, and 1840. [Facsimiles, ... edition published by W. Power& Co., Dublin.]*, Walton's Piano and Musical Instrument Galleries.

Camarena-Ibarrola, A. & Chávez, E., 2006. Identifying Music by Performances Using an Entropy Based Audio-Fingerprint. In *Mexican International Conference on Artificial Intelligence (MICAI)*.

Cambouropoulos, E., Crawford, T. & Iliopoulos, C.S., 2001. Pattern processing in melodic sequences: Challenges, caveats and prospects. *Computers and the Humanities*, 35(1), 9–21.

Carpenter, B., 2010. LingPipe: Download LingPipe Core Java Library. Available at: http://alias-i.com/lingpipe/web/download.html [Accessed April 28, 2010].

Chamberlin, D.D. & Boyce, R.F., 1974. SEQUEL: A structured English query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*. p. 264.

Chambers, J., 2010a. JC's ABC Tune Finder Homepage. *JC's ABC Tune Finder [tunefind] on trillian.mit.edu*. Available at: http://trillian.mit.edu/~jc/cgi/abc/tunefind [Accessed April 20, 2010].

Chambers, J., 2010b. O'Neill's Music of Ireland. *John Chambers' clone of the O'Neill's Project files and web pages*. Available at: http://trillian.mit.edu/%7Ejc/music/book/oneills/ [Accessed April 20, 2010].

Clausen, M. & Kurth, F., 2002. A unified approach to content-based and fault tolerant music identification. In Web Delivering of Music, 2002. WEDELMUSIC 2002. Proceedings. Second International Conference on. pp. 56-65.

Cronin, C., 1998. Concepts of melodic similarity in music-copyright infringement suits. *Melodic Similarity: Concepts, procedures and applications. MIT Press, Cambridge, Massachusetts*.

Cui, B. et al., 2008. Compacting music signatures for efficient music retrieval. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. pp. 229–240.

Damerau, F., 1964. A technique for computer detection and correction of spelling errors.

Dovey, M., 2001. A technique for "regular expression" style searching in polyphonic music. In Proc. ISMIR'2001. Citeseer.

Duggan, B., 2009. Machine Annotation of Traditional Irish Dance Music PhD Thesis.

Eerola, T. et al., 2000. Categorising Folk Melodies Using Similarity Ratings.

Emerick, C., 2003. Levenshtein Distance Algorithm: Java Implementation. Available at: http://www.merriampark.com/ldjava.htm [Accessed April 27, 2010].

Forbes, E., 1992. *Thayer's Life of Beethoven*, Princeton University Press.

Gatherer, N., 2009. Nigel Gatherer's ABC Collection Homepage. *Nigel Gatherer's ABC Collection*. Available at: http://www.nigelgatherer.com/tunes/abc.html [Accessed April 20, 2010].

Hamming, R., 1950. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2), 147-160.

Holzapfel, A. & Stylianou, Y., 2010. Similarity methods for computational ethnomusicology.

Hu, N. & Dannenberg, R.B., 2002. A comparison of melodic database retrieval techniques using sung queries. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. pp. 301–307.

Jaro, M.A., 1976. UNIMATCH Software System (No longer available).

Jaro, M.A., 1971. UNIMATCH: a computer system for generalized record linkage under conditions of uncertainty. In *Proceedings of the November 16-18, 1971, fall joint computer conference*. pp. 523–530.

Jaro, M., 1989. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406), 414-420.

Keith, J., 2010. The Session.org Homepage. *The Session*. Available at: http://www.thesession.org/ [Accessed April 20, 2010].

Larsen, G., 2003. *The Essential Guide to Irish Flute and Tin Whistle*, Mel Bay Publications, Inc.

Lavin, P., 2010. Irish Music Similarity Survey 2. Available at:

http://fluidsurveys.com/surveys/podge/irish-music-similarities-2-1/ [Accessed July 11, 2010].

Lemström, K. & Perttu, S., 2000. Semex-an efficient music retrieval prototype. In First International Symposium on Music Information Retrieval (ISMIR). Citeseer.

Lemström, K. & Ukkonen, E., 2000. Including interval encoding into edit distance based music comparison and retrieval. In Proceedings of the AISB'2000 Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science', Birmingham. Citeseer, pp. 53-60.

Lemström, K., Navarro, G. & Pinzon, Y., 2005. Practical algorithms for transposition-invariant string-matching. *Journal of Discrete Algorithms*, 3(2-4), 267–292.

Levenshtein, V., 1966. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet Physics-Doklady.

Likert, R., 1932. A technique for the measurement of attitudes. *Archives of Psychology. Vol*, 22(140), 55.

Lonelyhearts, M., 1978. The Session: Tunes - The Boys Of The Lough (reel). *The Session: Tunes - The Boys Of The Lough (reel)*. Available at: http://www.thesession.org/tunes/display/343 [Accessed May 5, 2010].

Mazzoni, D. & Dannenberg, R., 2001. Melody matching directly from audio. In 2nd Annual International Symposium on Music Information Retrieval. Citeseer, pp. 17-18.

McCullough, L., 1977. Style in Traditional Irish Music. *Ethnomusicology*, 21(1), 85-97.

Microsoft Corp., 2010. Microsoft DreamSpark. Available at: https://www.dreamspark.com/Default.aspx [Accessed April 20, 2010].

Moving Picture Experts Group, 1992. *MPEG. Coding of moving pictures and associated audio for digital storage media at up to 1.5 Mbit/s, part 3: Audio. International Standard IS 11172-3, ISO/IEC JTC1/SC29 WG11,*

Muellensiefen, D. & Frieler, K., 2003. 8 Cognitive Adequacy in the Measurement of Melodic Similarity: Algorithmic vs. Human Judgments.

Mullensiefen, D. & Frieler, K., 2007. Modelling expert's notions of melodic similarity. *MUSICAE SCIENTIAE*, 11(I), 183.

Norbeck, H., 1996. Henrik Norbeck's Abc Tunes Homepage. *Henrik Norbeck's Abc Tunes*. Available at: http://www.norbeck.nu/abc/ [Accessed April 20, 2010].

Oneill, 1980. *Waifs & Strays of Gaelic Melody* 2nd ed., Humanities Pr.

O'Neill, C.F., 1979. *O'Neill's Music of Ireland. Eighteen Hundred and fifty melodies. Airs, Jigs, Reels, Hornpipes, Long Dances, Marches, etc.*, Bronx NY: Daniel Michael Collins.

O'Neill, F., 1907. *The Dance Music of Ireland - 1001 Gems*, Chicago, USA.

O'Neill, F.O.&J., 1995. *O'Neill's 1001: The Dance Music of Ireland*, Walton's Mfg. Ltd.

Osna, 1999. *Osna*, Celtic Note.

Parsons, D., 1975. *The Directory of Tunes and Musical Themes* 1st ed., S. Brown.

Petrie, G., 2002. *The Petrie Collection of the Ancient Music of Ireland* 2nd ed., Cork University Press.

Pinto, A. & Haus, G., 2007. A novel XML music information retrieval method using graph invariants. *ACM Transactions on Information Systems*, 25(4), 19-es.

Rho, S. & Hwang, E., 2004. FMF (Fast Melody Finder): A Web-based Music Retrieval System. *Computer Music Modeling and Retrieval*, 351-388.

Sun Microsystems, 2010. NetBeans IDE Download. *NetBeans IDE Download*. Available at: http://netbeans.org/downloads/index.html [Accessed April 28, 2010].

Toiviainen, P. & Eerola, T., 2002. A computational model of melodic similarity based on multiple representations and self-organizing maps. In *Proceedings of the 7th International Conference on Music Perception and Cognition, Sydney*. pp. 236–239.

Walshaw, C., 1995. abcnotation.com Homepage. *Welcome to the home page at abcnotation.com. abc is a text based format for music notation, particularly popular for folk and traditional music.* Available at: http://abcnotation.com/ [Accessed April 20, 2010].

Wiggins, G.A., Lemstr\öm, K. & Meredith, D., 2002. SIA (M) ESE: An algorithm for transposition invariant, polyphonic, content-based music retrieval. In *3rd International Symposium on Music Information Retrieval (ISMIR 2002)*. pp. 13–17.

Winkler, W.E., 1999. The state of record linkage and current research problems. *Statistical Research Division, US Bureau of the Census, Wachington, DC*.

Winkler, W., 2006. Overview of record linkage and current research directions. *US Bureau of the Census Research Report*.

APPENDIX A – SURVEY PARTICIPANTS

Table 48: Panel of Experts in Irish traditional music

Name	Instrument played	Location
Hauke Steinberg	Flute and percussion	Germany
David Morrissey	Guitar and banjo	Kildare
Martin Preshaw	Uilleann pipes	Belfast
Daragh O'Reilly	Guitar and banjo	Mayo
Jose Manuel Fernandez Mateos	Bouzouki and percussion	Spain
Deirdre Smyth	Fiddle and flute	Dublin
Damian Werner	Flute	Hawaii
Paulo McNevin	Fiddle and flute	Dublin
Ray Dempsey	Button accordion	Waterford
Terry McGee	Flute	Australia
Pádhraic ó Súilleabheáin	Percussion	Kerry
Treasa Lavin	Whistle and piano	Mayo
Joe Brennan	Guitar	Cavan
Pauline Burke	Banjo	Dublin
Sara Cory	Fiddle	Chicago

Table 49: Panel of non-experts

Name	Location
Corinne Kingston Bageard	Illinois
Diarmuid Cooke	Dublin
Brian Duggan	Kerry
Martin Hughes	Louth
Joe Phelan	Dublin
John Golden	Mayo
Patrick Crowe	Dublin
John Breen	Sligo
Caroline Bemingham	England
Mark Bussell	North Carolina
Enora Senlanne	France
Richard Kinser	Texas
Terry Cosgrove	Clare
Clare Bassett	Dublin
Louisa Murphy	Cork

APPENDIX B - IRISH DANCE MUSIC SIMILARITIES SURVEY

A computer algorithm has picked the following tune parts as being somewhat similar **or** somewhat different.

All of the audio you are about to hear has been played by a computer. Please turn up the sound on your computer and play both audio samples in turn by clicking the triangular play button. Please listen to each sample as many times as you need to in order to make a decision.

There are no wrong answers, your opinion as a human is what is important.

Survey Start

Please enter your name:

Are you an expert in Irish Traditional Music?

○ Yes ○ No

Question 1

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 2

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 3

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 4

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 5

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 6

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 7

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 8

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 9

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

Question 10

Tune A  Tune B 

○ Very different ○ Different ○ I don't know ○ Similar ○ Very similar

APPENDIX C – SURVEY RESULTS

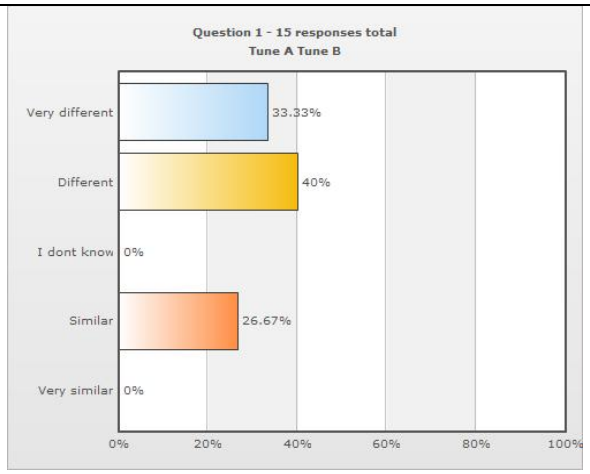


Figure 46: Experts responses to Question 1

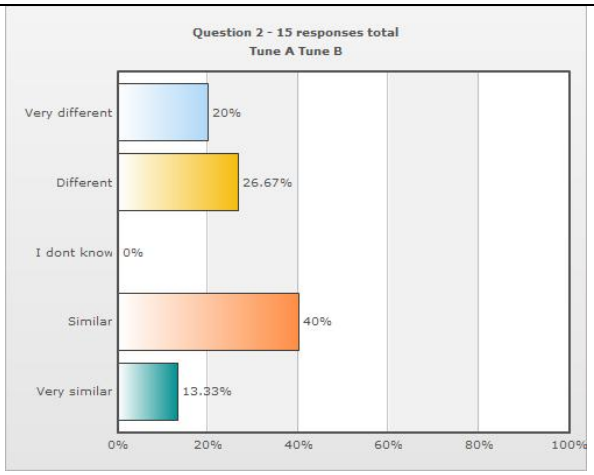


Figure 47: Experts responses to Question 2

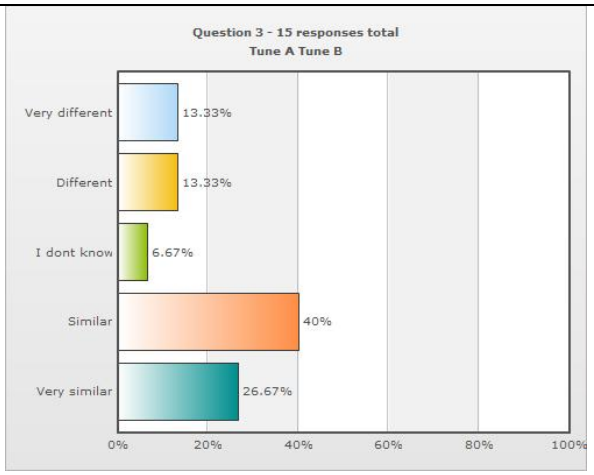


Figure 48: Experts responses to Question 3

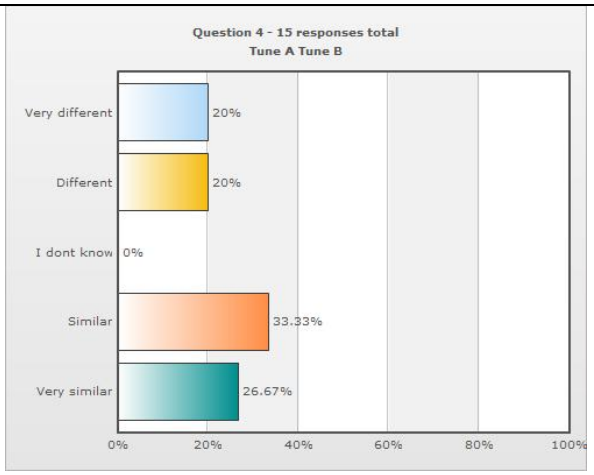


Figure 49: Experts responses to Question 4

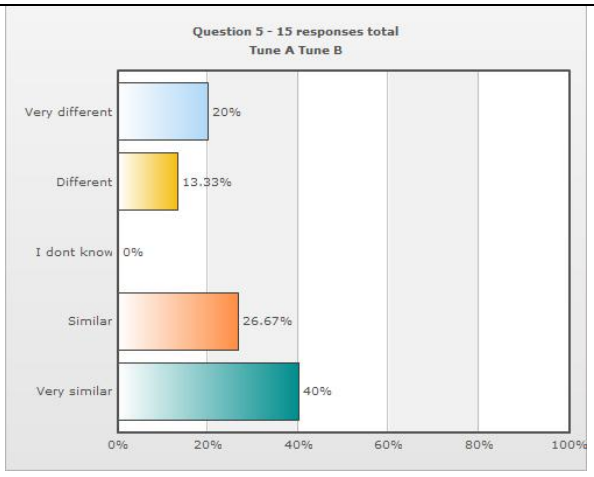


Figure 50: Experts responses to Question 5

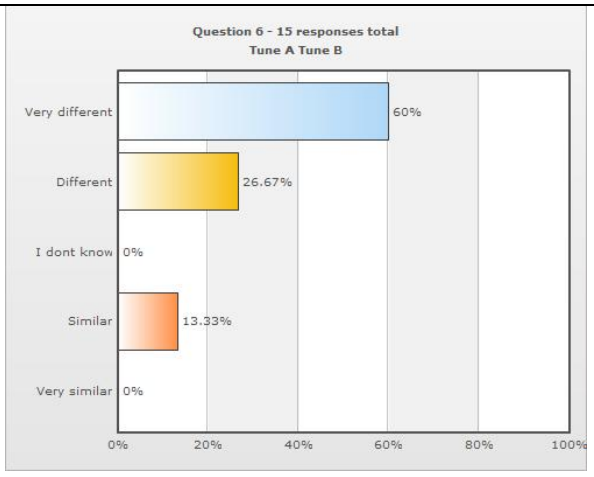


Figure 51: Experts responses to Question 6

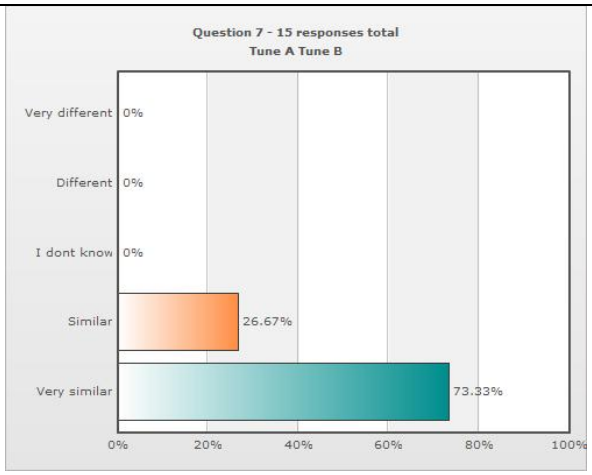


Figure 52: Experts responses to Question 7

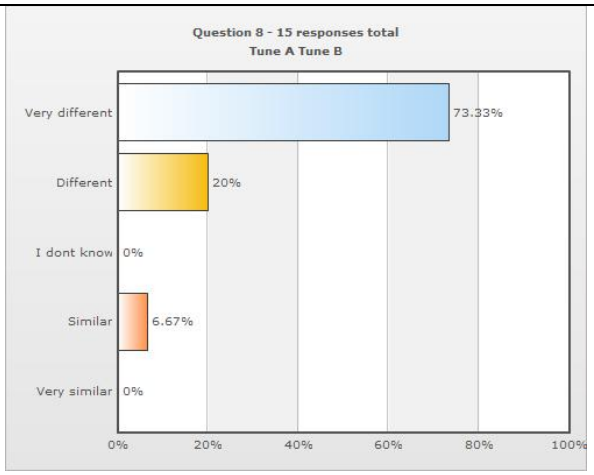


Figure 53: Experts responses to Question 8

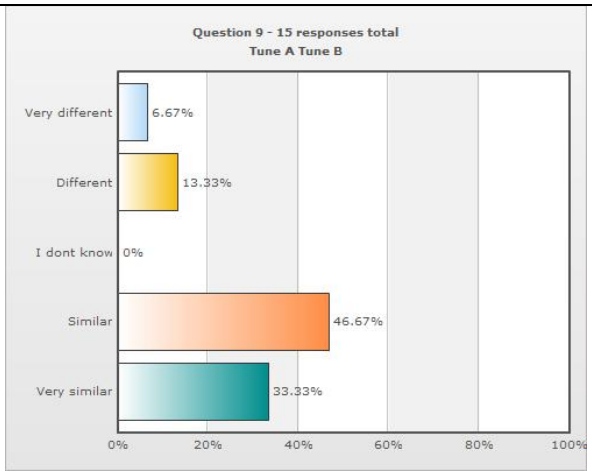


Figure 54: Experts responses to Question 9

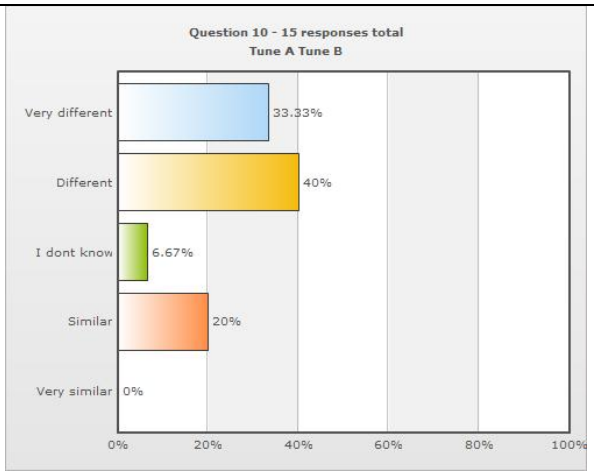


Figure 55: Experts response to Question 10

Non-experts responses to Questions 1- 10

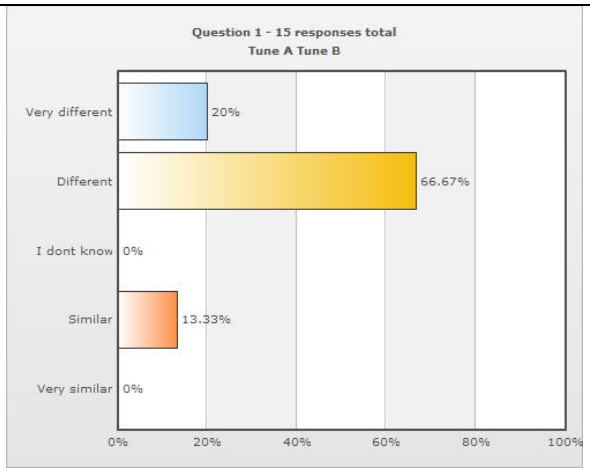


Figure 56: Non-experts responses to Question 1

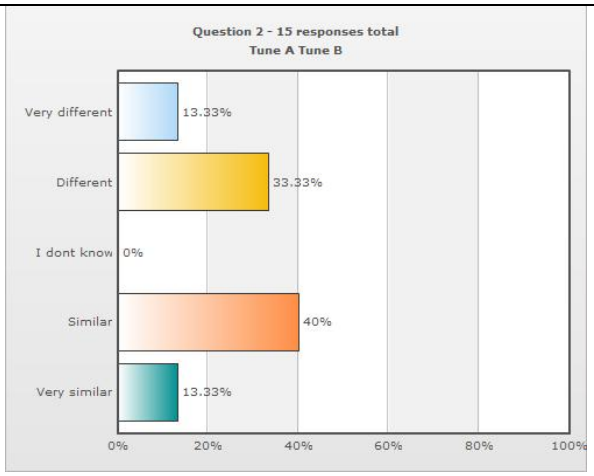


Figure 57: Non-experts responses to Question 2

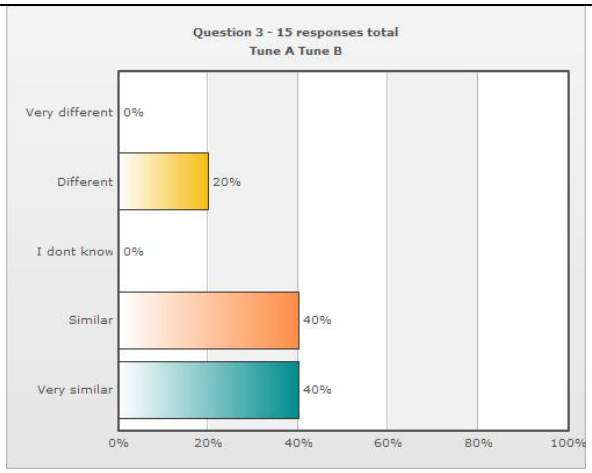


Figure 58: Non-experts responses to Question 3

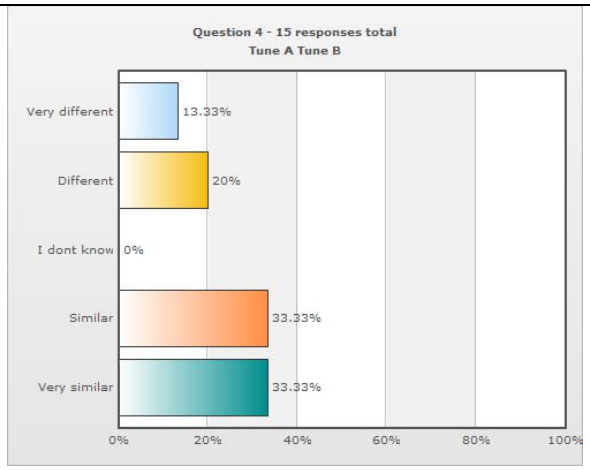


Figure 59: Non-experts responses to Question 4

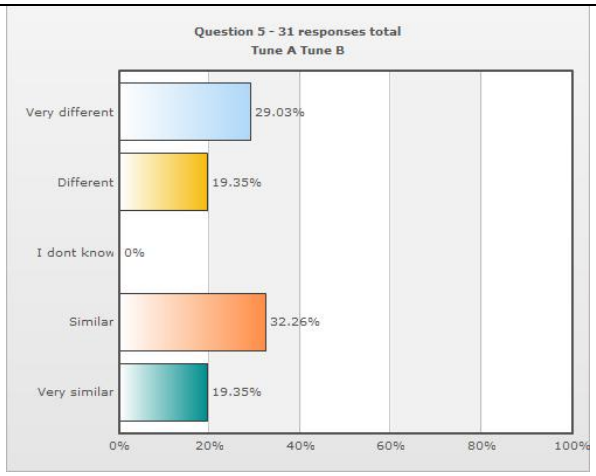


Figure 60: Non-experts responses to Question 5

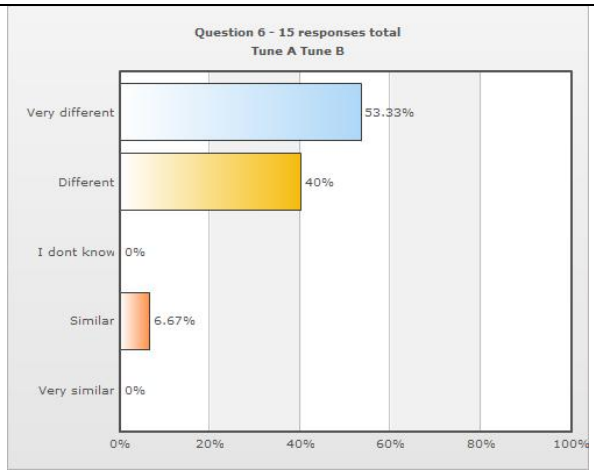


Figure 61: Non-experts responses to Question 6

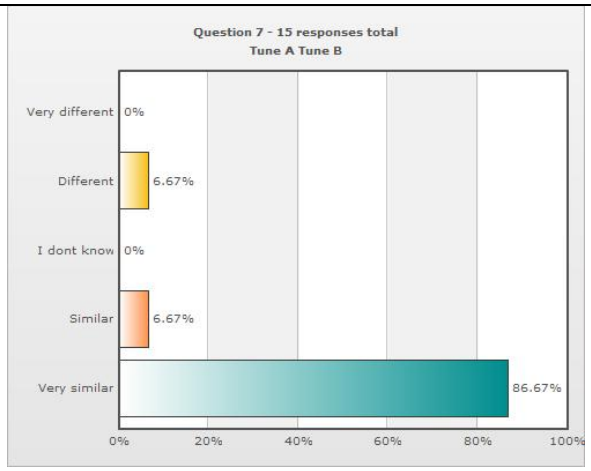


Figure 62: Non-experts responses to Question 7

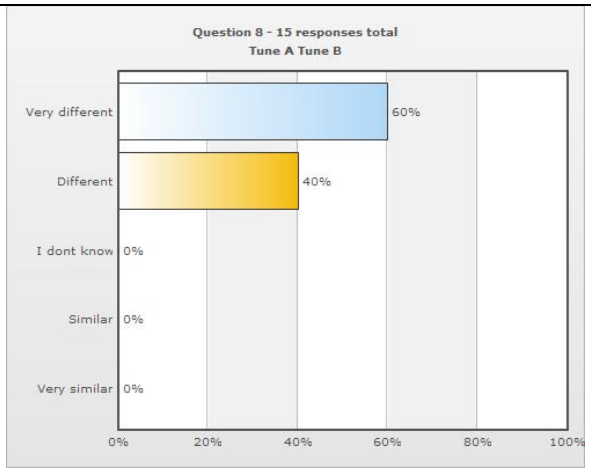


Figure 63: Non-experts responses to Question 8

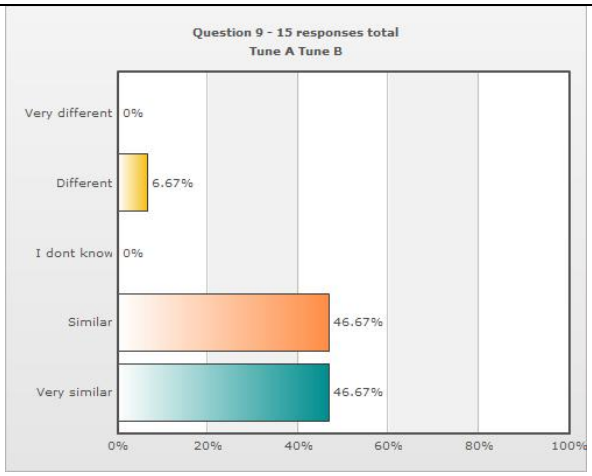


Figure 64: Non-experts responses to Question 9

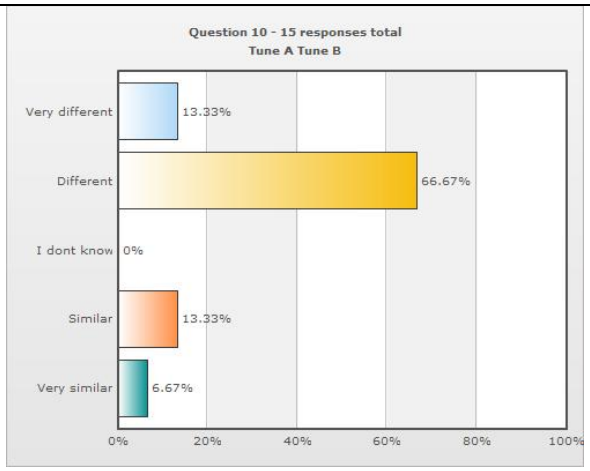


Figure 65: Non-experts responses to Question 10

Table 50: Overview of responses from all survey participants

Question:1	
Very different (1)	8 (26%)
Different (2)	16 (53%)
I dont know (3)	0 (0%)
Similar (4)	6 (20%)
Very similar (5)	0 (0%)
Total	30
Mean	2.13
Variance	1.09
Question:3	
Very different (1)	2 (6%)
Different (2)	5 (16%)
I dont know (3)	1 (3%)
Similar (4)	12 (40%)
Very similar (5)	10 (33%)
Total	30
Mean	3.77
Variance	1.63
Question:5	
Very different (1)	9 (30%)
Different (2)	6 (20%)
I dont know (3)	0 (0%)
Similar (4)	9 (30%)
Very similar (5)	6 (20%)

Question:2	
Very different (1)	5 (16%)
Different (2)	9 (30%)
I dont know (3)	0 (0%)
Similar (4)	12 (40%)
Very similar (5)	4 (13%)
Total	30
Mean	3.03
Variance	1.96
Question:4	
Very different (1)	5 (16%)
Different (2)	6 (20%)
I dont know (3)	0 (0%)
Similar (4)	10 (33%)
Very similar (5)	9 (30%)
Total	30
Mean	3.4
Variance	2.32
Question:6	
Very different (1)	17 (56%)
Different (2)	10 (33%)
I dont know (3)	0 (0%)
Similar (4)	3 (10%)
Very similar (5)	0 (0%)

Total	30
Mean	2.9
Variance	2.58
Question:7	
Very different (1)	0 (0%)
Different (2)	1 (3%)
I dont know (3)	0 (0%)
Similar (4)	5 (16%)
Very similar (5)	24 (80%)
Total	30
Mean	4.73
Variance	0.41
Question:9	
Very different (1)	1 (3%)
Different (2)	3 (10%)
I dont know (3)	0 (0%)
Similar (4)	14 (46%)
Very similar (5)	12 (40%)
Total	30
Mean	4.1
Variance	1.13

Total	30
Mean	1.63
Variance	0.86
Question:8	
Very different (1)	20 (66%)
Different (2)	9 (30%)
I dont know (3)	0 (0%)
Similar (4)	1 (3%)
Very similar (5)	0 (0%)
Total	30
Mean	1.4
Variance	0.46
Question:10	
Very different (1)	7 (23%)
Different (2)	16 (53%)
I dont know (3)	1 (3%)
Similar (4)	5 (16%)
Very similar (5)	1 (3%)
Total	30
Mean	2.23
Variance	1.22

Table 51: Overview of responses from expert survey participants

Question:1	
Very different (1)	5 (33%)
Different (2)	6 (40%)
I dont know (3)	0 (0%)
Similar (4)	4 (26%)
Very similar (5)	0 (0%)
Total	15
Mean	2.2
Variance	1.46
Question:3	
Very different (1)	2 (13%)
Different (2)	2 (13%)
I dont know (3)	1 (6%)
Similar (4)	6 (40%)
Very similar (5)	4 (26%)
Total	15
Mean	3.53

Question:2	
Very different (1)	3 (20%)
Different (2)	4 (26%)
I dont know (3)	0 (0%)
Similar (4)	6 (40%)
Very similar (5)	2 (13%)
Total	15
Mean	3
Variance	2.14
Question:4	
Very different (1)	3 (20%)
Different (2)	3 (20%)
I dont know (3)	0 (0%)
Similar (4)	5 (33%)
Very similar (5)	4 (26%)
Total	15
Mean	3.27

Variance	1.98
Question:5	
Very different (1)	3 (20%)
Different (2)	2 (13%)
I dont know (3)	0 (0%)
Similar (4)	4 (26%)
Very similar (5)	6 (40%)
Total	15
Mean	3.53
Variance	2.7
Question:7	
Very different (1)	0 (0%)
Different (2)	0 (0%)
I dont know (3)	0 (0%)
Similar (4)	4 (26%)
Very similar (5)	11 (73%)
Total	15
Mean	4.73
Variance	0.21
Question:9	
Very different (1)	1 (6%)
Different (2)	2 (13%)
I dont know (3)	0 (0%)
Similar (4)	7 (46%)
Very similar (5)	5 (33%)
Total	15
Mean	3.87
Variance	1.55

Variance	2.5
Question:6	
Very different (1)	9 (60%)
Different (2)	4 (26%)
I dont know (3)	0 (0%)
Similar (4)	2 (13%)
Very similar (5)	0 (0%)
Total	15
Mean	1.67
Variance	1.1
Question:8	
Very different (1)	11 (73%)
Different (2)	3 (20%)
I dont know (3)	0 (0%)
Similar (4)	1 (6%)
Very similar (5)	0 (0%)
Total	15
Mean	1.4
Variance	0.69
Question:10	
Very different (1)	5 (33%)
Different (2)	6 (40%)
I dont know (3)	1 (6%)
Similar (4)	3 (20%)
Very similar (5)	0 (0%)
Total	15
Mean	2.13
Variance	1.27

Table 52: Overview of responses from non-expert survey participants

Question:1	
Very different (1)	3 (20%)
Different (2)	10 (66%)
I dont know (3)	0 (0%)
Similar (4)	2 (13%)
Very similar (5)	0 (0%)
Total	15
Mean	2.07
Variance	0.78

Question:2	
Very different (1)	2 (13%)
Different (2)	5 (33%)
I dont know (3)	0 (0%)
Similar (4)	6 (40%)
Very similar (5)	2 (13%)
Total	15
Mean	3.07
Variance	1.92

Question:3	
Very different (1)	0 (0%)
Different (2)	3 (20%)
I dont know (3)	0 (0%)
Similar (4)	6 (40%)
Very similar (5)	6 (40%)
Total	15
Mean	4
Variance	1.29
Question:5	
Very different (1)	6 (40%)
Different (2)	4 (26%)
I dont know (3)	0 (0%)
Similar (4)	5 (33%)
Very similar (5)	0 (0%)
Total	15
Mean	2.27
Variance	1.78
Question:7	
Very different (1)	0 (0%)
Different (2)	1 (6%)
I dont know (3)	0 (0%)
Similar (4)	1 (6%)
Very similar (5)	13 (86%)
Total	15
Mean	4.73
Variance	0.64
Question:9	
Very different (1)	0 (0%)
Different (2)	1 (6%)
I dont know (3)	0 (0%)
Similar (4)	7 (46%)
Very similar (5)	7 (46%)
Total	15
Mean	4.33
Variance	0.67

Question:4	
Very different (1)	2 (13%)
Different (2)	3 (20%)
I dont know (3)	0 (0%)
Similar (4)	5 (33%)
Very similar (5)	5 (33%)
Total	15
Mean	3.53
Variance	2.27
Question:6	
Very different (1)	8 (53%)
Different (2)	6 (40%)
I dont know (3)	0 (0%)
Similar (4)	1 (6%)
Very similar (5)	0 (0%)
Total	15
Mean	1.6
Variance	0.69
Question:8	
Very different (1)	9 (60%)
Different (2)	6 (40%)
I dont know (3)	0 (0%)
Similar (4)	0 (0%)
Very similar (5)	0 (0%)
Total	15
Mean	1.4
Variance	0.26
Question:10	
Very different (1)	2 (13%)
Different (2)	10 (66%)
I dont know (3)	0 (0%)
Similar (4)	2 (13%)
Very similar (5)	1 (6%)
Total	15
Mean	2.33
Variance	1.24

APPENDIX D – PROGRAMMING CODE

Table 53: Code snippet of the Semex implementation in C# based on Dr. Bryan Duggan's Java implementation

```
        /// <summary>
        /// gets the similarity of the two strings using Semex distance.
        /// </summary>
        /// <param name="firstWord"></param>
        /// <param name="secondWord"></param>
        /// <returns>a value between 0-1 of the similarity</returns>
        public override double GetSimilarity(string firstWord, string
secondWord) {
            if ((firstWord != null) && (secondWord != null)) {

                // Convert strings to arrays of midi notes
                int[] pattern = notesToMidiArray(firstWord);
                int[] text = notesToMidiArray(secondWord);

                double Semex = calculateSemex(pattern, text);

                if (pattern.Length >= text.Length)
                {
                    return 1-(Semex / text.Length);
                }
                else {
                    return 1-(Semex / pattern.Length);
                }
            }
            return defaultMismatchScore;
        }

        /// <summary>
        /// gets the un-normalised similarity measure of the metric for the
given strings.</summary>
        /// <param name="firstWord"></param>
        /// <param name="secondWord"></param>
        /// <returns> returns the score of the similarity measure (un-
normalised)</returns>
        public override double GetUnnormalisedSimilarity(string firstWord,
string secondWord) {
            return GetSimilarity(firstWord, secondWord);
        }

        /// <summary>
        /// Converts a string of notes to an array of midi notes
        /// </summary>
        private int[] notesToMidiArray(string input2)
        {
            string control = "CDEFGAB";
            string input = "";
            input2 = input2.ToUpper();

            // clean input
            for (int i = 0; i < input2.Length; i++)
            {
                if (control.IndexOf(input2[i]) != -1)
                {
                    input += input2[i];
```

```
                }
            }

            int[] tmp = new int[input.Length];

            // convert to midi notes
            for (int i = 0; i < input.Length; i++)
            {
                if (control.IndexOf(input[i]) != -1)
                {
                    tmp[i] = 60 + control.IndexOf(input[i]);
                }
            }
            return tmp;
        }

        /// <summary>
        /// Calculates the Semex edit distance between two int arrays, pattern
and text
        /// </summary>
        public double calculateSemex(int[] pattern, int[] text)
        {
            int pLength = pattern.Length;
            int tLength = text.Length;
            int difference = 0;

            //Console.Out.Write("pLength:" + pLength + " tLength:" + tLength);

            int sc;

            if (pLength == 0)
            {
                return -1;
            }
            if (tLength == 0)
            {
                return -1;
            }

            int[][] d = new int[pLength + 1][];

            // Initialise all rows to be zero instead of null based.
            for (int i = 0; i < pLength + 1; i++)
            {
                d[i] = new int[tLength + 1];
            }

            // Initialise the first row
            for (int i = 0; i < tLength + 1; i++)
            {
                d[0][i] = 0;
            }
            // Now make the first col = 1,2,3,4,5,6
            for (int i = 0; i < pLength + 1; i++)
            {
                d[i][0] = i;
            }


            for (int i = 1; i <= pLength; i++)
            {
                sc = pattern[i - 1];
```

```
                for (int j = 1; j <= tLength; j++)
                {
                    int v = d[i - 1][j - 1];
                    if (j - 2 < 0 || i - 2 < 0)
                    {
                        difference = 1;
                    }
                    else if ((text[j - 1] - text[j - 2]) != (pattern[i - 1] -
pattern[i - 2]))
                    {
                        difference = 1;
                    }
                    else
                    {
                        difference = 0;
                    }
                    d[i][j] = Math.Min(Math.Min(d[i - 1][j] + 1, d[i][j - 1] +
1), v + difference);
                }
            }


            int[] lastRow = d[pLength];
            int min = int.MaxValue;
            for (int i = 1; i < tLength + 1; i++)
            {
                int c = lastRow[i];
                if (c < min)
                {
                    min = c;
                }
            }
            return min;
        }
```

Table 54: Breathnach MIC Implementation in C#

```
        /// <summary>
        /// gets the similarity of the two strings using MIC distance.
        /// </summary>
        /// <param name="firstWord"></param>
        /// <returns>a value between 0-1 of the similarity</returns>
        public override string GetSimilarity(string firstWord)
        {
            if ((firstWord != null))
            {
                return calculateBreathnach(firstWord);
            }
            else
            {
                return "";
            }

        }

        /// <summary>
        /// gets the Parsons code for a string.
        /// </summary>
        private string calculateBreathnach(string input2)
        {
            try
```

```
            {
                input2 = input2.ToUpper();
                string control = "CDEFGAB";
                StringBuilder input = new StringBuilder();

                // clean input
                for (int i = 0; i < input2.Length; i++)
                {
                    if (control.IndexOf(input2[i]) != -1)
                    {
                        input.Append(input2[i]);
                    }
                }

                String key = input[input.Length - 1].ToString();
                StringBuilder temp = new StringBuilder();
                int char1, interval, fundamental;
                fundamental = control.IndexOf(key);
                for (int i = 0; i < input.Length; i++)
                {
                    try
                    {
                        char1 = control.IndexOf(input[i]);
                        interval = (char1 - fundamental + 1);
                        if (interval < 1)
                        {
                            interval += 7;
                        }
                        temp.Append(interval);
                    }
                    catch (Exception e)
                    {
                        //Console.Out.WriteLine(e.ToString());
                    }
                }
                return temp.ToString();
            }
            catch
            {
                return "Error!";
            }
        }
```

Table 55: Parsons Code Implementation in C#

```
        /// <summary>
        /// gets the similarity of the two strings using Parsons distance.
        /// </summary>
        /// <param name="firstWord"></param>
        /// <returns>a value between 0-1 of the similarity</returns>
        public override string GetSimilarity(string firstWord)
        {
            if (firstWord != null)
            {
                return calculateParsons(firstWord);
            }
            else
            {
                return "";
            }
        }
```

```
        /// <summary>
        /// gets the Parsons code for a string.
        /// </summary>
        private string calculateParsons(string input2)
        {
            try
            {
                input2 = input2.ToUpper();
                string control = "CDEFGAB";
                string input = "";

                // clean input
                for (int i = 0; i < input2.Length; i++)
                {
                    if (control.IndexOf(input2[i]) != -1)
                    {
                        input += input2[i];
                    }
                }

                string temp = "*";
                int loc, loc2 = 0;
                for (int i = 0; i < input.Length - 1; i++)
                {
                    loc = control.IndexOf(input[i]);
                    loc2 = control.IndexOf(input[i + 1]);
                    if (loc > loc2) { temp += "D"; }
                    else if (loc == loc2) { temp += "R"; }
                    else if (loc < loc2) { temp += "U"; }
                }
                return temp;
            }
            catch
            {
                return "Error!";
            }
        }
```

Table 56: Standard Deviation Function in C# based on a C# version freely available online

```
        /// <summary>
        /// gets the stdev of the four values passed to it.
        /// </summary>
        /// <param name="firstValue"></param>
        /// <param name="secondValue"></param>
        /// <param name="thirdValue"></param>
        /// <param name="fourthValue"></param>
        /// <returns>a value between 0-1 of the similarity</returns>
        public override double GetSimilarity(double firstValue, double
secondValue, double thirdValue, double fourthValue)
        {

            ArrayList rankList = new ArrayList();
            rankList.Add(firstValue);
            rankList.Add(secondValue);
            rankList.Add(thirdValue);
            rankList.Add(fourthValue);
```

```
            return StandardDeviation(rankList);

        }

        /// <summary>
        /// gets the normalised rank of the four rank values passed to it. Also
need the total count of records (the highest rank)
        /// </summary>
        /// <param name="firstValue"></param>
        /// <param name="secondValue"></param>
        /// <param name="thirdValue"></param>
        /// <param name="fourthValue"></param>
        /// <param name="count"></param>
        /// <returns>a value between 0-1 of the normalised rank</returns>
        public double GetNormalisedRank(int firstValue, int secondValue, int
thirdValue, int fourthValue, int count)
        {
            // need four ranks and the total count of records (highest rank) to
normalise
            int sum = firstValue + secondValue + thirdValue + fourthValue;
            double normalisedRank = 1.0 - ((sum - 4.0) / (count * 4.0));
            return normalisedRank;
        }

        ///<Summary>
        ///Calculates standard deviation of numbers in an ArrayList
        ///</Summary>
        public static double StandardDeviation(ArrayList num)
        {
            double SumOfSqrs = 0;
            double avg = Average(num);
            for (int i = 0; i < num.Count; i++)
            {
                SumOfSqrs += Math.Pow(((double)num[i] - avg), 2);
            }
            double n = (double)num.Count;
            return Math.Sqrt(SumOfSqrs / (n - 1));
        }

        ///<Summary>
        ///Calculates average of numbers of integer data type in an ArrayList
        ///</Summary>
        public static double Average(ArrayList num)
        {
            double sum = 0.0;
            for (int i = 0; i < num.Count; i++)
            {
                sum += (double)num[i];
            }
            double avg = sum / System.Convert.ToDouble(num.Count);

            return avg;
        }
```

Table 57: TextFunctions String Metrics Assembly

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlTypes;
using SimMetricsMetricUtilities;
```

```
using Microsoft.SqlServer.Server;

    public class StringMetrics
    {

        private static readonly Levenstein _Levenstein;
        private static readonly Jaro _Jaro;
        private static readonly JaroWinkler _JaroWinkler;
        private static readonly Semex _Semex;
        private static readonly Breathnach _Breathnach;
        private static readonly Parsons _Parsons;
        private static readonly stdevmusic _stdevmusic;

        static StringMetrics()
        {
            _Levenstein = new Levenstein();
            _Jaro = new Jaro();
            _JaroWinkler = new JaroWinkler();
            _Semex = new Semex();
            _Breathnach = new Breathnach();
            _Parsons = new Parsons();
            _stdevmusic = new stdevmusic();

        }


        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlDouble Levenstein(SqlString firstWord, SqlString
secondWord)
        {
            if (firstWord.IsNull || secondWord.IsNull)
                return 0;

            return new SqlDouble(_Levenstein.GetSimilarity(firstWord.Value,
secondWord.Value));
        }

        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlDouble Jaro(SqlString firstWord, SqlString secondWord)
        {
            if (firstWord.IsNull || secondWord.IsNull)
                return 0;

            return new SqlDouble(_Jaro.GetSimilarity(firstWord.Value,
secondWord.Value));
        }


        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlDouble JaroWinkler(SqlString firstWord, SqlString
secondWord)
        {
            if (firstWord.IsNull || secondWord.IsNull)
                return 0;

            return new SqlDouble(_JaroWinkler.GetSimilarity(firstWord.Value,
secondWord.Value));
        }
```

```
        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlDouble Semex(SqlString firstWord, SqlString secondWord)
        {
            if (firstWord.IsNull || secondWord.IsNull)
                return 0;

            return new SqlDouble(_Semex.GetSimilarity(firstWord.Value,
secondWord.Value));
        }

        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlString Breathnach(SqlString firstWord)
        {
            if (firstWord.IsNull)
                return "";

            return new SqlString(_Breathnach.GetSimilarity(firstWord.Value));
        }

        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true, DataAccess=DataAccessKind.Read)]
        public static SqlDouble BreathnachRank(SqlString firstWord, SqlString
secondWord)
        {
            if (firstWord.IsNull || secondWord.IsNull)
                return 0;

            return new SqlDouble(_Breathnach.GetSimilarity(firstWord.Value,
secondWord.Value));
        }

        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlString Parsons(SqlString firstWord)
        {
            if (firstWord.IsNull)
                return "";

            return new SqlString(_Parsons.GetSimilarity(firstWord.Value));
        }

        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlDouble stdevmusic(SqlDouble firstValue, SqlDouble
secondValue, SqlDouble thirdValue, SqlDouble fourthValue)
        {
            if (firstValue.IsNull || secondValue.IsNull || thirdValue.IsNull ||
fourthValue.IsNull)
                return 0.0;

            return new SqlDouble(_stdevmusic.GetSimilarity(firstValue.Value,
secondValue.Value, thirdValue.Value, fourthValue.Value));
        }

        [Microsoft.SqlServer.Server.SqlFunction(IsDeterministic = true,
IsPrecise = true)]
        public static SqlDouble normalisedRank(int firstValue, int secondValue,
int thirdValue, int fourthValue, int count)
        {
```

```
            if (firstValue.Equals(null) || secondValue.Equals(null) ||
thirdValue.Equals(null) || fourthValue.Equals(null) || count.Equals(null))
                return 0.0;

            return new SqlDouble(_stdevmusic.GetNormalisedRank(firstValue,
secondValue, thirdValue, fourthValue, count));
        }
    }
```

Table 58: SQL to install custom string distance functions in MS SQL 2008

```
DROP FUNCTION Levenstein
GO

DROP FUNCTION NeedlemanWunch
GO

DROP FUNCTION SmithWaterman
GO

DROP FUNCTION SmithWatermanGotoh
GO

DROP FUNCTION SmithWatermanGotohWindowedAffine
GO

DROP FUNCTION Jaro
GO

DROP FUNCTION JaroWinkler
GO

DROP FUNCTION ChapmanLengthDeviation
GO

DROP FUNCTION ChapmanMeanLength
GO

DROP FUNCTION QGramsDistance
GO

DROP FUNCTION BlockDistance
GO

DROP FUNCTION CosineSimilarity
GO

DROP FUNCTION DiceSimilarity
GO

DROP FUNCTION EuclideanDistance
GO

DROP FUNCTION JaccardSimilarity
GO

DROP FUNCTION MatchingCoefficient
GO

DROP FUNCTION MongeElkan
```

```
GO

DROP FUNCTION OverlapCoefficient
GO

DROP FUNCTION Semex
GO

DROP FUNCTION Parsons
GO

DROP FUNCTION Breathnach
GO

DROP FUNCTION BreathnachRank
GO

DROP FUNCTION stdevmusic
GO

DROP FUNCTION NormalisedRank
GO

DROP ASSEMBLY [TextFunctions]
GO

CREATE ASSEMBLY [TextFunctions]
AUTHORIZATION [dbo]
FROM
'C:\bin\TextFunctions\TextFunctions\bin\Release\TextFunctions.dll'
WITH PERMISSION_SET = EXTERNAL_ACCESS
GO

CREATE FUNCTION Levenstein(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.Levenstein
GO

CREATE FUNCTION NeedlemanWunch(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.NeedlemanWunch
GO

CREATE FUNCTION SmithWaterman(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.SmithWaterman
GO

CREATE FUNCTION SmithWatermanGotoh(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.SmithWatermanGotoh
GO

CREATE FUNCTION SmithWatermanGotohWindowedAffine(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.SmithWatermanGotohWindowedAffine
GO
```

```
CREATE FUNCTION Jaro(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.Jaro
GO

CREATE FUNCTION JaroWinkler(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.JaroWinkler
GO

CREATE FUNCTION ChapmanLengthDeviation(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.ChapmanLengthDeviation
GO

CREATE FUNCTION ChapmanMeanLength(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.ChapmanMeanLength
GO

CREATE FUNCTION QGramsDistance(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.QGramsDistance
GO

CREATE FUNCTION BlockDistance(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.BlockDistance
GO

CREATE FUNCTION CosineSimilarity(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.CosineSimilarity
GO

CREATE FUNCTION DiceSimilarity(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.DiceSimilarity
GO

CREATE FUNCTION EuclideanDistance(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.EuclideanDistance
GO

CREATE FUNCTION JaccardSimilarity(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.JaccardSimilarity
GO

CREATE FUNCTION MatchingCoefficient(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.MatchingCoefficient
```

```
GO

CREATE FUNCTION MongeElkan(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.MongeElkan
GO

CREATE FUNCTION OverlapCoefficient(@firstword
NVARCHAR(255),@secondword NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.OverlapCoefficient
GO

CREATE FUNCTION Semex(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.Semex
GO

CREATE FUNCTION Breathnach(@firstword NVARCHAR(255))
RETURNS nvarchar(255) EXTERNAL NAME
TextFunctions.StringMetrics.Breathnach
GO

CREATE FUNCTION BreathnachRank(@firstword NVARCHAR(255),@secondword
NVARCHAR(255))
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.BreathnachRank
GO

CREATE FUNCTION Parsons(@firstword NVARCHAR(255))
RETURNS nvarchar(255) EXTERNAL NAME
TextFunctions.StringMetrics.Parsons
GO

CREATE FUNCTION stdevmusic(@firstValue float, @secondValue float,
@thirdvalue float, @fourthValue float)
RETURNS float EXTERNAL NAME TextFunctions.StringMetrics.stdevmusic
GO

CREATE FUNCTION NormalisedRank(@firstValue int, @secondValue int,
@thirdvalue int, @fourthValue int, @count int)
RETURNS float EXTERNAL NAME
TextFunctions.StringMetrics.normalisedRank
GO
```

Table 59: getRanks Stored Procedure

```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[getRanks]    Script Date:
06/29/2010 14:30:05 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:        Padraic Lavin
-- Create date: 2010
-- Description:   Return ranks
```

```
-- ============================================
CREATE PROCEDURE [dbo].[getRanks]
      -- Add the parameters for the stored procedure here
      @Notes nvarchar(255) = ''
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

      -- Declare some Variables
      DECLARE @parsons nvarchar(255) = [Test].[dbo].Parsons(@notes)
      DECLARE @breathnach nvarchar(255) =
[Test].[dbo].Breathnach(@notes)
      DECLARE @rowID float
      DECLARE @MaxRank float
      DECLARE @prowID float
      DECLARE @PMaxRank float
      DECLARE @count int

      -- Create temp table for Breathnach Rank and populate it
      SELECT
      ID,
      dbo.Breathnach(NOTES) as MIC,
      dbo.Parsons(NOTES) as PIC,
      row_number() over (order by dbo.Breathnach(NOTES) asc) as
rowID,
      (row_number() over (order by dbo.Breathnach(NOTES)))/1.0 as
MICScore,
      (row_number() over (order by dbo.Parsons(NOTES)))/1.0 as
PICScore
      into #TEMP
      from corpus
      order by MIC;

      -- Find nearest match for @notes - Breathnach
      Select top 1 @rowID = MICScore from #TEMP where
dbo.Breathnach(@notes) <= MIC order by MIC asc
      Select @MaxRank = MAX(MICScore) from #TEMP

      -- Find nearest match for @notes - Parsons
      Select top 1 @prowID = PICScore from #TEMP where
dbo.Parsons(@notes) <= PIC order by PIC asc
      Select @PMaxRank = MAX(PICScore) from #TEMP

      Update #TEMP set MICScore = 1-((abs(MICScore -
@rowID))/@MaxRank)
      Update #TEMP set PICScore = 1-((abs(PICScore -
@prowID))/@PMaxRank)

    -- Insert statements for procedure here

      select [Test].[dbo].Corpus.ID, NAME, PART,
[Test].[dbo].Semex(@Notes, dbo.corpus.NOTES) as Semex,
RANK() OVER(ORDER BY [Test].[dbo].Semex(@Notes, dbo.corpus.[NOTES])
DESC) AS [SemexRank],
[Test].[dbo].JaroWinkler(@Notes, dbo.corpus.NOTES) as Jaro,
RANK() OVER(ORDER BY [Test].[dbo].JaroWinkler(@Notes,
dbo.corpus.[NOTES]) DESC) AS [JaroRank],
[Test].[dbo].JaroWinkler(@parsons,
[Test].[dbo].Parsons(dbo.corpus.NOTES)) as Parsons,
```

```
RANK() OVER(ORDER BY #TEMP.PICScore DESC) AS [ParsonsRank],
#TEMP.MICScore,
RANK() OVER(ORDER BY #TEMP.MICScore DESC) AS [MICRank],
notes, tunekey, measure, #TEMP.MIC

into #TEMP3

from dbo.corpus
left join #TEMP
on [Test].[dbo].Corpus.ID = #TEMP.ID
order by MICRank asc, SemexRank asc, JaroRank asc, ParsonsRank asc

select @count = COUNT(ID) from #TEMP3

select ID, NAME, NOTES, TUNEKEY, MEASURE,
[Test].[dbo].stdevmusic(SemexRank, JaroRank, ParsonsRank, MICRank) as
stdev, [Test].[dbo].normalisedRank(SemexRank, JaroRank, ParsonsRank,
MICRank, @count) as NRank
from #TEMP3
order by NRank desc, stdev asc
END
```

Table 60: getRanksID Stored Procedure

```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[getRanksID]    Script Date:
06/29/2010 14:31:36 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:        <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   <Description,,>
-- =============================================
CREATE PROCEDURE [dbo].[getRanksID]
      -- Add the parameters for the stored procedure here
      @ID int
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

      -- Declare some Variables
      DECLARE @notes nvarchar(255)

      select @notes = NOTES from [Test].[dbo].corpus where ID = @ID

      DECLARE @parsons nvarchar(255) = [Test].[dbo].Parsons(@notes)
      DECLARE @breathnach nvarchar(255) =
[Test].[dbo].Breathnach(@notes)
      DECLARE @rowID float
      DECLARE @MaxRank float
      DECLARE @prowID float
      DECLARE @PMaxRank float
      DECLARE @count int
```

```
      -- Create temp table for Breathnach Rank and populate it
      SELECT
      ID,
      dbo.Breathnach(NOTES) as MIC,
      dbo.Parsons(NOTES) as PIC,
      row_number() over (order by dbo.Breathnach(NOTES) asc) as
rowID,
      (row_number() over (order by dbo.Breathnach(NOTES)))/1.0 as
MICScore,
      (row_number() over (order by dbo.Parsons(NOTES)))/1.0 as
PICScore
      into #TEMP
      from corpus
      order by MIC;

      -- Find nearest match for @notes - Breathnach
      Select top 1 @rowID = MICScore from #TEMP where
dbo.Breathnach(@notes) <= MIC order by MIC asc
      Select @MaxRank = MAX(MICScore) from #TEMP

      -- Find nearest match for @notes - Parsons
      Select top 1 @prowID = PICScore from #TEMP where
dbo.Parsons(@notes) <= PIC order by PIC asc
      Select @PMaxRank = MAX(PICScore) from #TEMP

      Update #TEMP set MICScore = 1-((abs(MICScore -
@rowID))/@MaxRank)
      Update #TEMP set PICScore = 1-((abs(PICScore -
@prowID))/@PMaxRank)

    -- Insert statements for procedure here

      select @ID as A_ID, [Test].[dbo].Corpus.ID as B_ID,
[Test].[dbo].Semex(@Notes, dbo.corpus.NOTES) as Semex,
RANK() OVER(ORDER BY [Test].[dbo].Semex(@Notes, dbo.corpus.[NOTES])
DESC) AS [SemexRank],
[Test].[dbo].JaroWinkler(@Notes, dbo.corpus.NOTES) as Jaro,
RANK() OVER(ORDER BY [Test].[dbo].JaroWinkler(@Notes,
dbo.corpus.[NOTES]) DESC) AS [JaroRank],
[Test].[dbo].JaroWinkler(@parsons,
[Test].[dbo].Parsons(dbo.corpus.NOTES)) as Parsons,
RANK() OVER(ORDER BY #TEMP.PICScore DESC) AS [ParsonsRank],
#TEMP.MICScore,
RANK() OVER(ORDER BY #TEMP.MICScore DESC) AS [MICRank],
notes, tunekey, measure, #TEMP.MIC

into #TEMP3

from dbo.corpus
left join #TEMP
on [Test].[dbo].Corpus.ID = #TEMP.ID
order by MICRank asc, SemexRank asc, JaroRank asc, ParsonsRank asc

select @count = COUNT(A_ID) from #TEMP3

select A_ID, B_ID, [Test].[dbo].stdevmusic(SemexRank, JaroRank,
ParsonsRank, MICRank) as stdev,
[Test].[dbo].normalisedRank(SemexRank, JaroRank, ParsonsRank,
MICRank, @count) as NRank
from #TEMP3
```

```
order by NRank desc, stdev asc
END
```

Table 61: calculateMatrix Stored Procedure

```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[calculateMatrix]    Script
Date: 06/29/2010 14:32:29 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:        <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   <Description,,>
-- =============================================
CREATE PROCEDURE [dbo].[calculateMatrix]
AS

      -- SET NOCOUNT ON added to prevent extra result sets from
      -- interfering with SELECT statements.
      SET NOCOUNT ON;

      -- Declare some Variables
      DECLARE @notes nvarchar(255)
      DECLARE @corpusID int
      DECLARE     @return_value int

      DECLARE tune_cursor CURSOR FOR SELECT cast([ID] as int) as ID,
NOTES FROM [Test].[dbo].[corpus] where (ID >= 8353 and ID <= 20297)
order by ID asc

      OPEN tune_cursor
      -- Perform the first fetch.
      FETCH NEXT FROM tune_cursor into @corpusID, @notes

      BEGIN

      -- Check @@FETCH_STATUS to see if there are any more rows to
fetch.
      WHILE @@FETCH_STATUS =0
      BEGIN
      --Select @corpusID, @notes

      -- ***************************************************
      INSERT dbo.matrix (A_ID, B_ID, STDEV, NRank)
      EXEC  @return_value = [dbo].[getRanksID]
            @ID = @corpusID

--***************************************************

FETCH NEXT FROM tune_cursor into @corpusID, @notes

END

CLOSE tune_cursor
DEALLOCATE tune_cursor
END
```