Composing music by constraint satisfaction: using human composed music to improve the quality of computer generated sequences

A dissertation submitted in partial fulfilment
of the requirements for the Open University's
Master of Science Degree
in Computing for Commerce and Industry

Nicholas Hindle
(T5156277)

9 March 2008

Word Count: 14,878

Preface

I would like to thank Christian Schulte of the KTH Royal Institute of Technology in Sweden, for his assistance with my implementation of GECODE algorithms.

My thanks also go to those who helped in proof reading my work, and especially to my Open University supervisor, Dr. M.A.Wermelinger for his guidance and support throughout the project. I would also like to thank my wife for her assistance, and her patience in listening to some of the rather curious sounds that were produced by my computer during the project.

Table of Contents

List of Figures

ABSTRACT

Composing music was attempted in experiments using the first generations of digital computer as early as 1949 (Wikipedia$_1$, 2008). Although the capability and complexity of computer technology has increased considerably in recent history, advances in computer music are restricted by a number of limitations, especially how to measure the quality of the music that the computer has composed.

This research seeks to analyse the effects of using a database of human composed music as reference data with the objective of improving the musical quality of computer music.

A popular method used for computer-assisted music composition is *constraint programming*. This alternative to conventional programming techniques operates in a declarative manner, where the programmer declares the problem by stating an algorithm and the constraint satisfaction system provides solutions to the problem. The programmer does not need in-depth knowledge of how the solutions are created in order to produce successful solutions.

In order to assess the quality of computer music, investigations into techniques for comparing fragments of music were carried out, and a technique called *melodic contour matching* was adopted as a concept for matching music phrases.

A practical implementation of contour matching was built into an experimental prototype software tool. The tool produced music sequences that were

programmatically compared to a database of human composed folk music, and the parameters of the composition program were adjusted according to the results of matching, allowing the effects of the feedback to influence subsequent compositions.

The music sequences produced by the constraint satisfaction system alone proved to be of limited quality, but after applying the feedback from the contour matching process, changes in the structure of the composed sequences were observed and the melodic structure of the generated music began to show some affinity with that of the human composed music.

1 INTRODUCTION

Computers have been used to assist with the composition of music as long as digital computer technology has existed. The prospect of computers creating high quality music is one that still provokes considerable interest. In a recent *Guardian Unlimited* article, in considering whether computers could write a chart-topper, McCready (2006) suggested, "80% of all pop songs that had ever been hits shared a relatively small number of underlying structures. There was, then, something about a successful song that was non-random and measurable".

The techniques and methods available for computer assisted composition range from purpose built sequencing systems, to elaborate artificial intelligence systems that are popular in the academic world. Purpose built tools include "jMusic", created by Sorensen and Brown (2000), which is a suite of classes in the computer language Java™ that provides a music composition language based on Common Music Notation and Hierarchical Music Specification Language (HMSL) standards. "Cubase" is a commercially available music sequencing software package based on proprietary compositional methods. Wikipedia$_2$ (2007) claims that the algorithms used in Cubase have proved so successful they are incorporated into virtually all major sequencing software packages.

There are three key groups of techniques used in research into computer-assisted composition research.

- Stochastic methods are based largely on conventional programming tools, and

operate by applying strict mathematical processes to music theory rules;

- More contemporary work employs the use of artificial intelligence, which aims to model the human thought processes involved in the composition process;
- The third technique employs *constraint programming*, due to its ability to produce imprecise or approximate results, which is a key requirement of music composition.

1.1 Constraint Programming

Constraint programming is a relatively new approach to computer programming that operates in a *declarative* way; the programmer states the problem and supplies the necessary variables, and the constraint satisfaction program generates solutions based on internal mechanisms. The solutions produced by a constraint program can include approximate or inexact results, a task that is notoriously difficult for conventional imperative programming techniques.

Leler (1988) illustrates how a program written in an imperative computer language is a step-by-step procedure, where the programmer solves problems by manually constructing specific executable algorithms. Problem groups which are to be solved by an imperative program require the programmer to anticipate each problem to be solved and define explicit decision points that determine which algorithm should be applied.

In contrast, a constraint programmer states a set of relations between a set of objects and the constraint-satisfaction system attempts to find solutions that satisfy these

relations, without the programmer being involved in how the solutions are derived. The programmer can thus create solutions without an in-depth understanding of the underlying logic.

Leler summarises the concepts of constraint programming with the following characteristics:

- A constraint expresses a desired relationship among one or more objects.
- A constraint language describes the objects and the constraints.
- A constraint satisfaction system finds solutions to constraint language programs.
- Constraint satisfaction uses problem-solving methods, called 'constraint satisfaction techniques' to find the values of the objects that will make the relationships true.

An example illustrates the different approaches taken by imperative and constraint programs. The algorithm used by an imperative program to convert Fahrenheit (F) to Celsius(C) is given by `(F-32) x 5/9`. Given a value of F, this imperative algorithm could determine the value of C.

However, given a value of C, the algorithm would be unable to determine the value of F. Hence the program would require a branch statement to determine the execution path of the program, depending on which value is supplied. A constraint program would use an algorithm which would initially appear to be identical; `C = (F-32) x 5/9`, but this differs from the imperative version as it defines the relationship between the two objects F and C. Given a value for F *or* C, the constraint satisfaction system might use algebraic rules to determine the required value for the other.

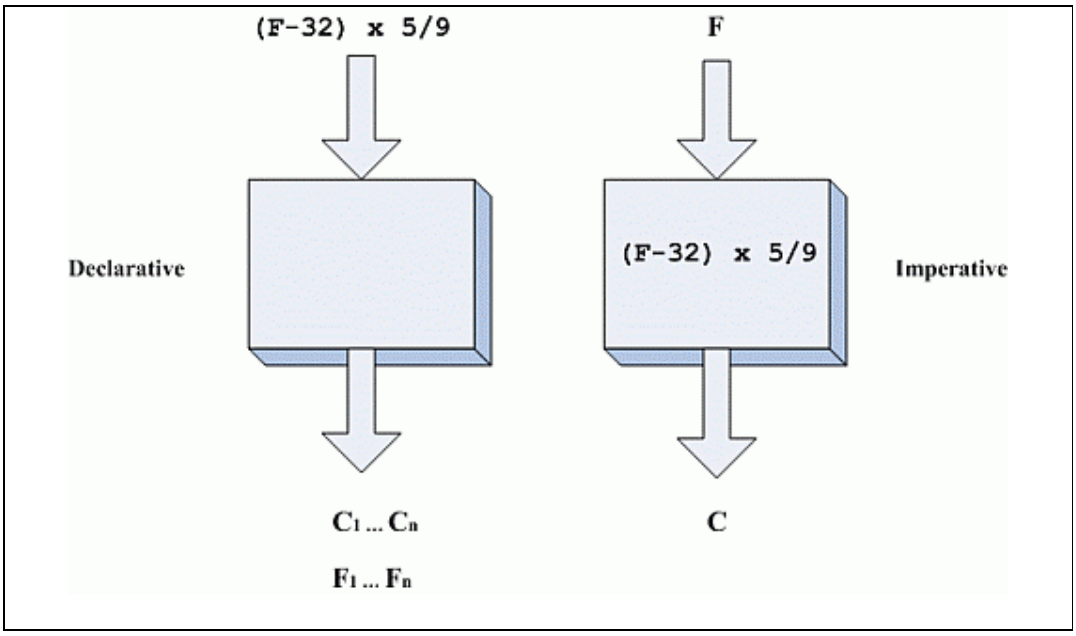

Figure 1.1 Declarative and Imperative programming

Figure 1.1 shows how declarative programming uses indeterminate methods (depicted by the empty process box) to create solutions that may produce all the valid values for C given F, and all the valid values for F given C.

The constraint satisfaction problem (CSP) operates in a *domain*, so for a variable called *x*, the domain of x denoted by D(*x*), represents a set of values that can be assigned to *x*. The formal definition of this is $D(x) = \{d_1,\ldots d_m\}$, and in the context of music generation, *x* represents a single note in the solution, and the values of *d* are available notes that can be assigned to the note *x*. Because a whole sequence of notes is required in the solution, the general definition is $X = x_1, x_2,\ldots, x_m$ with respective domains $D = D(x_1), D(x_2),\ldots D(x_n)$. Note that curly braces "{ }" are often omitted to simplify notation.

A technique used by many constraint satisfaction tools is "branching". This is a general algorithm that systematically enumerates all candidate solutions (in the search

space) and discards those that are not consistent with the required solution. The branching procedure recursively produces a search tree of subsets of candidates, resulting in one or more subsets of consistent solutions.

To illustrate, a simple constraint problem called *belong* states that a set of 3 integers e.g. {1,4,6} must be among those in the domain, which is the range of integers between 1 and 10. The branching mechanism will generate all the possible combinations of three numbers, then divide these into two groups, one of which contains sets that are likely to succeed and one that contains sets that do not. The latter group can be discarded and the first group is further divided until the only group left is the one containing successful solutions. A set of solutions for this example might include {1,2,4},{2,4,6},{5,8,9}

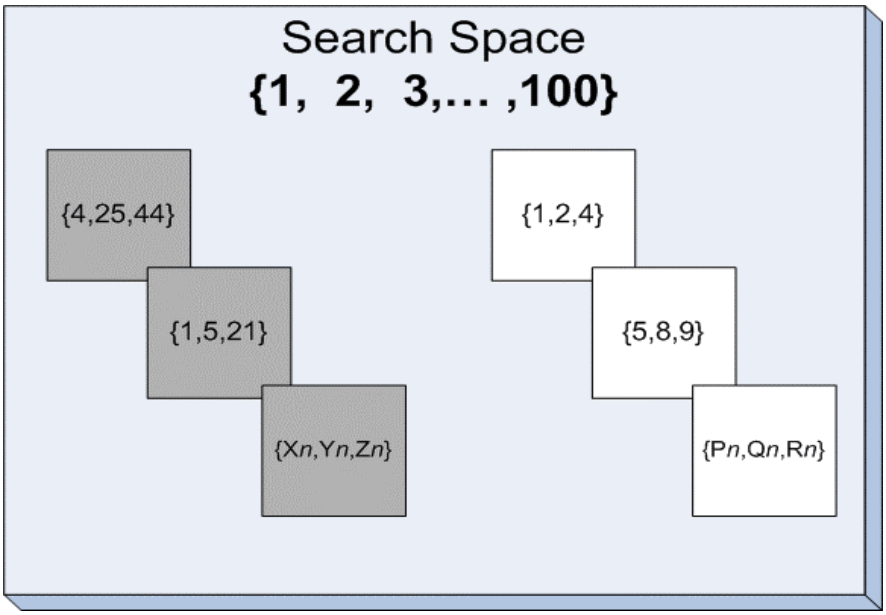

Figure 1.2 The "belong" constraint

Figure 1.2 shows how the branching process of a constraint satisfaction system produces solutions for the "belong" constraint. The branching process produces

groups of integers of the required length and asserts whether they are consistent with the required solution. In this example, consistent solutions are shown in white, and *P*, *Q* and *R* represent all the consistent solutions not explicitly illustrated. Inconsistent solutions are shown in grey, and *X*, *Y* and *Z* represent all the inconsistent solutions that are not explicitly illustrated.

1.2 The Quality of Music

While there is no effort involved in you or I deciding whether a piece of music sounds good, it is very difficult to model this cognitive action with computer based analytical methods. A potential solution to this problem is to create computer simulations based on models of human creative thought based processes. However this approach requires in-depth understanding of the thought processes involved and belongs more to the realm of psychological research, which is outside the scope of this work.
An alternative and more practical approach is to use a database of human composed music to provide the composition system with a source of reference data. In order to evaluate the validity of this approach, this dissertation aims to seek answers to the following questions:

- Can generated music sequences be refined as a result of matching with existing music sequences stored in a reference database?
- Does the refinement process improve the quality of the generated compositions?
- Does the use of databases make composition easier and improve matching performance?

- Can the reference data be used to identify common patterns in different musical styles?
- Could this approach be complemented by learning systems to generate parameters for a given style?

The music database is a vital and critical part of the analysis. In fact, Cope (2001) suggests "experiments in Musical Intelligence rely almost completely on its database for creating new compositions". In order to be able to carry out effective music sequence comparison, it is necessary to store the sequence data in a format that supports matching. Many music databases contain binary data, which store the actual waveforms of sounds that have been captured and digitised. Music data in this form can be used for accurate audio reproduction, but matching melodies created by a computer with such data is extremely complex. A popular method is to convert the music into tokenised data strings that represent standard codes, such as MIDI (musical instrument digital interface) sequences.

1.3 Music Sequence Matching

Computer created music can be assessed simply by listening. However, this task could become somewhat tedious, assuming a large number of fragments are to be assessed, and maintaining an objective assessment may prove difficult. A mechanism called *contour matching* has proved effective in providing quantitative comparison of music sequences in work by Uitdenbogerd and Zobel (1998) and Downie (2000).

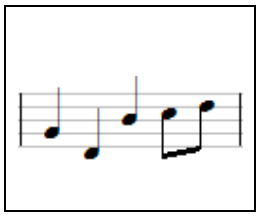

Figure 1.3 Sample music fragment

Figure 1.3 shows a bar of standard notation music, known as an *n-gram*. This is simply a small fragment of musical notes of known length. There are a number of ways of defining where an n-gram starts and ends and how long it should be, many of which are based on complex rules. For the purposes of this research, an n-gram based on a single bar proved to be effective, so all further references to the n-gram can be assumed to follow this pattern.

D	**U**	**U**	**U**

Figure 1.4 Sample sequence contour

Figure 1.4 shows a sequence contour that was created using the method used by Uitdenbogerd and Zobel. This is useful to illustrate how the characters U, D, S (up, down or the same) represent the changes in direction of notes. Note that a contour is relative, so an identical fragment played in different octaves will exhibit the same contour and can be recognised as the same tune by the listener.

1.4 Objectives

The objective of this research is to analyse the effects of combining constraint-based music composition techniques with contour matching against a database of human composed music, and to determine whether any improvement in quality of the computer composed music can be gained as a result of this combination of techniques.

The information gained from the literature review in chapter two of the dissertation is used to determine an appropriate selection of methods for analysing constraint-based music composition and music sequence matching. Chapter three presents the chosen experimental research method and introduces the design and development of a functional prototype that is used to provide the data for the primary analysis. Chapter four discusses the implementation of the experimental prototype, showing how the discrete functional parts of the system were combined and how the data was collected for analysis. The results of the experiments are presented and analysed in chapter five, and conclusions drawn from the analysis are discussed in chapter six, along with suggestions for further research.

2 LITERATURE SURVEY

It is assumed that the reader understands the essential elements of music, including key signatures, musical time constructs, and how notes and rests are grouped in order to represent a structured piece of music. A basic understanding of database technology, such as tables, rows and columns is also assumed.

A diverse range of computer-based methods has been applied to the essentially human, thought-based practice of music composition. The scope of the field is wide, and hence this review is split into a chronological progression, from discussions on music perception and early works on the construction of music, to more contemporary studies employing techniques such as artificial intelligence, constraint satisfaction, and music sequence databases.

2.1 Music Perception

The way that humans perceive music is a particularly important consideration when designing automated music generation systems. In music, an interval is the distance from one note to another, and Longuet-Higgins (1993) suggests that the commonest intervals occurring in traditional melodies are "diatonic", where the interval is less than six - intervals of six (Diabolic), or greater (Chromatic), are relatively rare. He does not appear to support this proposition with any evidence, but observation of samples of music in the Western[1] music tradition does in fact corroborate this theory.

[1] See glossary

An implicit property of music is that the tonal identity of a note cannot be established until the following note is heard. Longuet-Higgins supports this notion by comparing the problems posed by melodic perception to those encountered in the perception of speech. For example, the words “here” and “hear” are completely different in meaning but they sound exactly alike when spoken. The meaning of a word is derived from the context of the phrase in which they appear. One might dispute that the context of a musical note is in fact a lot easier to determine than the hundreds of ambiguous words in Western languages, but the comparison is interesting all the same. Longuet-Higgins uses the remainder of the text to discuss the design of a computer program that applies a number of absolute methods to a collection of fundamental rules such as those discussed above, to produce music. The program requires initialisation by a performer, who has to establish the initial tempo and number of notes in a bar. The formal approach taken by this system appears somewhat inflexible, which is confirmed by Longuet-Higgins’ concluding comments; “The domain of competence of the program is, of course, very restricted”.

2.2 Construction of Simple Sequences

Schoenberg (1967) describes the smallest unit of musical structure, the *phrase,* as “a kind of musical molecule consisting of integrated musical events, possessing certain completeness, and well adapted to combination with similar units”. Schoenberg suggests one could sing a phrase in a single breath. The concept of the phrase is useful throughout this project, which attempts to determine how to compose such phrases and potentially combine them into larger, more complete musical pieces. Schoenberg recognizes that “phrase endings may be marked by a combination of distinguishing

features, such as rhythmic reduction, melodic relaxation though a drop in pitch, the use of smaller intervals and new notes; or by any other suitable differentiation". I suggest this clearly illustrates the intrinsic complexity in even the simplest of musical fragments; a human composer *knows* if a phrase ending "sounds" right, but it is clear that this subtle detail presents a challenge for computers. Schoenberg describes musical *sentences*, which consist of collections of phrases. He states a key characteristic of the organisation of phrases within a sentence is repetition - "Intelligibility in music seems to be impossible without repetition" and that "repetition without variation can easily produce monotony"; he would probably be very disappointed with the monotonous nature of many of the contemporary offerings. It is interesting to observe that Schoenberg states "the beginning of the sentence determines the construction of the continuation". While this is not a strict rule for musical composition, it is unusual for music not to follow this fundamental imperative. Schoenberg uses the notion of the *contour* in his discussions around cadence and other musical sentence constructions. The contour is a concept still in general use by many authors of works related to music composition, and is a fundamental principle for this project.

2.3 Computer Assisted Music Composition

The fundamental concepts of musical composition are provided by Cope (2001), where the basic building blocks of music are described as the *recombination of pitches and durations*. Cope suggests that the recombination of larger groupings of pitches and durations form the basis of musical styles. Hofstadter (2001) contributes to the work by Cope by asking the "big" question: Will a computer program ever write beautiful music? He describes music as a language of emotions, and until programs have emotions as complex as ours, it is impossible for a program to produce anything beautiful. Hofstadter suggests that a music box might soon be available which could produce pieces of similar standard to Chopin or Bach, but then counters this by suggesting that a computer would need to live a human life and encounter the experiences that generate emotions within us in order to produce emotive and hence beautiful music. In contrast to Hofstadter's statement, I suggest that it is possible for a computer program to produce beautiful music using algorithmic or heuristic methods, rather than by emotion.

Cope (2001) describes "recombinant music" as "the identification of recurrent structures of various sorts in a composer's output, and the re-using of those structures in new arrangements, so as to construct a new piece *in the same style*". There is no reason why computer based music generation cannot combine heuristic composition methods with recombination techniques to reach the goal of original and beautiful music creation. Cope introduces the concept of *signatures,* which are contiguous note patterns that occur in two or more works of a single composer, and are critical to the recognition of musical style. A signature nominally comprises between two and five

beats (four to ten melodic notes), and usually consists of composites of melody, harmony and rhythm. Cope states that signatures typically occur between four and ten times in any given work, however, this appears to be a rather restrictive limit; smaller note patterns are bound to exist more frequently in a piece of music but would not necessarily determine the structure or identity of the piece as a whole. The text continues to discuss the topic of pattern matching which can be used to identify signatures in a sequence. Cope states that exact matches are not required in the context of music pattern matching because "precisely repeating sequences are the exception rather than the norm". I suggest that the key to this discussion is that it is vitally important to identify similar patterns of music and not exact sequences.

In his discussions around musical structure, Cope introduces the process of detecting thematic boundaries, an important concept for the identification of fragments of music sequences in larger compositions. Repeating or contrasting thematic ideas and their repetitions are delineated by sectional boundaries. However, one might argue that thematic pattern analysis is a very complex approach that exceeds the majority of requirements for identifying sequences. A simpler approach that appears to be more commonly used is the "n-gram", which is a sequence of known length. Assuming the basic structure of the piece of music is known, the starting point for extracting n-grams is normally the first note in a bar, and the length is either fixed or determined by the location of a particular boundary marker such as the next bar line on the stave.

In the subsequent chapter on databases, Cope points out that quantizing musical data from MIDI (musical instrument digital interface) files for use as a reference database is a time-consuming and difficult task. However, his approach to modelling the internal format of MIDI appears to add complexity to this task by requiring transformation and storing of fixed data elements such as *on-time, pitch,* and *channel*

number. As Cope suggests, choosing works for a database is a fairly obvious process; the widest range of sources will produce the widest variety of compositions. Reflecting on experiments in Musical Intelligence, Cope observes that a composition system must "rely almost completely on its database for creating new compositions". However, in his statement "a single error in the database can have serious consequences in the output music", Cope somewhat contradicts his earlier claims that exact sequences are not important, so in the context of original music composition, some "errors" in the data may in fact be an advantage.

Brooks *et al.* (1993) discuss two of the major components of human thought processes - induction and deduction. The authors illustrate how computers are adept at deductive reasoning, but are somewhat limited in their ability for inductive reasoning. In order to counter this limitation, the authors suggest the approach of having the machine analyze a sample of acceptable compositions and applying the conclusions in order to derive new, original compositions. Brooks identifies a problem with this approach in that the synthetic note sequences may not belong to the class of acceptable melodies from which they were derived. While this may conflict with his theoretical expectations, it is arguably not a problem if the aim is to generate original compositions. Brooks discusses another problem pertinent to this work, where the samples produced are too short to allow effective analysis. There is a minimum realistic size for a sequence; anything smaller would be likely to occur in too many existing pieces of music to allow effective selection and analysis.

Rader (1993) coins a description of the goal of his work on computer assisted music composition; "The goal here is not to make aesthetically perfect music, but to make it

indistinguishable to the human ear from human produced music". Arguably, computer generated music which is indistinguishable from human composed music could be described as aesthetically perfect; this is the goal of the human composer.

2.4 Music Composition and Artificial Intelligence

Balaban *et al.* (1992) provide a synopsis of music in the field of Artificial Intelligence research. In the introductory chapters of this work, a key statement states that "Computers are physical systems which are simultaneously symbol manipulators and pattern recognisers". This confirms the suitability of using computers to generate music by manipulating musical symbols, and by using pattern recognition methods to refine the generated music sequences. In subsequent discussions, Balaban suggests that computers provide "a demonstration of how it is possible for a physical system to carry out mental pursuits". This seems a rather optimistic statement; simulation of mental pursuits is certainly feasible but, by definition, mental pursuits are thought processes, which are arguably beyond the capability of current computer technology.

Kugel (1992) describes how a computer program that purports to describe how Mozart wrote piano sonatas could automatically compose piano sonatas in Mozart's style, even including some sonatas that Mozart never actually wrote. There is no evidence that this was achieved, and in fact the author soon countermands this theory and suggests, quite rightly, that computations alone might not be powerful enough to describe all of musical thinking. The author discusses how the application of computational theories can be supported by the use of irrational numbers to enable them to deal with problems relating to the mental world. He describes how "un-

computations" are introduced into the conceptual toolkit for computational musicology, leading to more powerful ways of using computer programs. The term *limiting computations* is used to identify this approach, and describes how such limiting computations can be run on computers, much like ordinary computations. In the context of musicology, it is suggested that limiting computations are able to produce results that cannot necessarily be proven, which is an unusual but positive attribute in the inexact science of computer music composition. A quote from Einstein provides a fitting conclusion to this piece of work; "a scientific theory should be as simple as possible. But not simpler."

2.5 Constraint Programming in Computer Assisted Music Composition

A considerable amount of research has proved that constraint programming can provide the potential for generating original music sequences. A library of general constraint satisfaction problems is available on a website called "CSPLib" (Gent and Walsh, 2006). This provides a useful background to many of the constraints discussed in this dissertation, and the combinatorial nature of many constraint algorithms provides basic problem solving for many sequence based problems, including music sequence generation.

A whole paper by van Hoeve (2001) is dedicated to the *all-different* constraint, which provides a very comprehensive analysis of this fundamental but powerful algorithm. The definition of this CSP is AllDifferent(v,o) where v and o are arrays whose range is $1..n$ and the constraints $v[i] + o[i] \neq v[j] + o[j]$ hold for all $1 = i < j = n$.
van Hoeve *et al.* (2006) further discuss a number of sequencing constraints, beginning

with the fundamental *among* constraint. This simply states that the variables are assigned a value from a specific set. This *sequence* constraint states that a number of consecutive variables is assigned to at least *min* and at most *max* values from a set of available values (the set of values may be a solution from *among* constraint). The discussions contained within this work are highly theoretical and not presented in the context of any particular real world application, but are interesting as a foundation for building further sequencing constraints.

Anders (2007) has produced a considerable amount of work on the subject of computer-assisted composition by the use of constraint solving tools, and uses the phrase "music constraint programming". Anders has produced a composition system called *Strasheela* that is based on constraint programming methods. The foundation constraint satisfaction problem used in Strasheela is the *all-interval* series.
This constraint is based on the conjunction of two simple rules; one ensures that all pitches are pair-wise distinct, and the other ensures that all intervals between successive pitches are pair-wise distinct.

Pitch	**0**	**11**	**1**	**10**	**2**	**9**	**3**	**8**	**4**	**7**	**5**	**6**
Interval		**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

Figure 2.1 An all-interval solution

A solution for twelve pitches is shown in Figure 2.1 which shows that each interval of between one and eleven semitones occurs only once.

While Strasheela appears to provide rich functionality for composing original music scores, it requires the user to have a thorough understanding of constraint programming and is thus out of reach for many composers and musicians who are not necessarily computer programmers. Nevertheless, this contemporary work provides a valuable background into the concepts related to constraint-based music composition.

Patchet and Roy (2004) are prolific writers in the area of music composition by constraint programming, and describe a number of fundamental constraints. Before describing the application of constraints, they illustrate two basic ingredients that apply to music generation: “the issue in music selection is to find the right combination between [repetition and surprise, to] provide users with … items they do not know, but probably will like”. Obviously this comment is in the context of music selection but applies equally to music composition. The *similar* constraint, as the name suggests, states that items in a given range are successively similar to each other. This ability to constrain values to follow similar sequences extends the concept of recombination discussed by Cope earlier in this chapter. Patchet and Roy describe *all-different*, another constraint commonly used for solving sequencing problems. This CSP requires all the pairs of variables in the solution to be different. The solution

to each all-different problem will be the first feasible sequence on which the constraint holds.

At this point it is important to note that the solutions produced by any constraint satisfaction problem are *consistent*, and given the same constraint definition and variables will produce the same collection of solutions. *All-interval, all-different* and *similar*, are no different; the finite set of results from each solution represents a predictable collection of sequences. Further processing is required to reduce determinism within the solution set. To achieve this, the alternatives are to perform some arbitrary ordering of the domain variables prior to execution of the constraint solver, or to add further constraints to the basic algorithm to facilitate more complex modelling. Smith (2003) discusses how variable ordering can improve the performance of sequencing constraints, whereas Zimmerman (2004) recommends the use of randomness "to close the degree of non-determinism".

Patchet and Roy extend the discussion on constraint satisfaction in discussing cardinality constraints. These are useful in imposing properties on sets of items, a technique that might be used to ensure that notes in the solution belong to a given musical key. The implementation of these constraints is complex, and may result in long execution times. However, this is a relatively simple problem that may be easier to implement using imperative methods, which can be applied to the constraint variables prior to running the constraint satisfaction program, or to the solution on completion.

Zimmerman (2004) presents some interesting concepts for modelling music using computers. He observes that while there are "intentions of music that exist already

before or during the composition process, for instance [including] emotions", "a computer cannot, obviously, be *spontaneously* forced to a certain composition by impressions or emotions". This is the fundamental challenge that faces all experiments in computer based music generation. Zimmerman labels such external influences as *intentions* and the result of using such intentions in composition is the *intentional structure*. He suggests a solution to including intentions in computer-assisted composition is the use of a database of 'canned' music, but discards the idea due to problems anticipated with time coordination and "attrition of musical effects". However, the addition of human influence to a computerised solution for a creative problem such as music composition is unlikely to be surpassed by "composition from scratch" as he advocates. In the section headed *Newness,* Zimmerman states "an intelligent, distributed concept of randomness is needed that doses the degree of non-determinism". While there must be some degree of randomness in computer-assisted composition, the use of a database of human composed music adds the essential intelligence element that is otherwise extremely difficult to model. The remainder of the work describes the application of sequencing constraints in a system called "Coppelia", and how the user provides the parameters for the application's search strategies. Like many constraint based music generation solutions, Zimmerman's user is required to provide a large amount of information for the composition system to use, and uses terminology that belongs neither in the realm of pure music composition or constraint variable planning, therefore it would appear that considerable expertise is required in order to obtain worthwhile results.

Truchet and Codognet (2004) follow a similar path to Zimmerman in their work. The authors provide a description of how computer music composition is an ideal

constraint satisfaction problem (CSP). They suggest that standard textbook rules for music composition such as “no parallel fifth” and “opposite motion between two voices” are “written in a declarative way”. This affirms the affinity between music composition and the declarative way that constraint programming is defined.

The work initially visits fourteen existing musical constraint satisfaction problems, which have been stated by a number of authors in the field. These include harmony generation (chords), rhythmical structures, and melodies. In the melodies section, they introduce the concept of the *gesture,* which is a musical interval or short series of intervals, which relates closely to the *signature* introduced by Cope (2001). Gestures are applied to an initial fixed note to form a melody. Cope suggests *all-interval* is a trivial solution as the requirement is simple; re-organisation of a sequence of integers. While the authors claim they do not want trivial solutions, it is a useful solution and hence this approach has been used extensively to solve problems in computer assisted music composition. To affirm the limitations of constraint programming discussed earlier in this dissertation, Truchet and Codognet assert that given the same fundamental properties, the CSP provides predictable solutions.

Codognet proposes heuristic extensions to the basic constraint search mechanism called adaptive search. The basic algorithm starts with a random configuration of values, then selects an adequate neighbour from the search space, and moves to the next best candidate. This process continues until a satisfactory solution is found. The adaptive search approach modifies the CSP paradigm from “problem -> solution” to “partial information -> make suggestions”. This is implemented in a CSP environment called “OMClouds” which has been implemented in the software package “OpenMusic” which is presented as a “visual programming language”. While this encapsulation assists in using the built-in constraints for music composition, it still

requires considerable expertise and understanding of the concepts involved in constraint programming. The authors admit limitations in terms of performance of the system but do not provide any quantitative measurements. The performance is simply described as “adequate” for the purposes of their application. A final comment does raise some concern however; a “musical expert” is needed to further extend the research, suggesting this work is not much more than an initial investigation.

Although somewhat outdated, Moorer (1972) offers some interesting ideas in his work on Music and Computer Composition. He observes that in a single composition, the harmony and rhythm may vary drastically. While this makes sequence matching more difficult, it is a hurdle that must be overcome in research into computer music composition. While a particular piece of music may contain great variation, it can be argued that each and every piece of music must belong to a given style or genre. Moorer suggests we consider “what it is that makes a random sequence of notes a “good” or a “bad” melody”. While a random sequence of notes may in fact be a “good” melody, the crux of music composition is not to create random notes, but to use knowledge and experience to create “good” music. The author stresses the importance of rhythm in music by defining the term *periodicity*; “Periodicity is very deeply impressed on people of all cultures”. Moorer draws from other work to describe how the melody [in a piece of music] “may be organized into sections that repeat in exact or in transposed form”. This detail is most important when deriving rules for original music composition, and for selection and matching sequences in existing music. Also important is his observation that a listener can identify a popular melody very quickly from hearing only a small portion of the piece. In his experiments in composition, Moorer determines the structure of a piece of music

based on groups, and using the power of two as a basis for the length of the groups and the overall piece. This mathematical approach seems very limited, and in a review of Moorer's work, Smoliar (1972) points out that models for simulating human composition models were available for over 25 years prior to the date of the paper. However, Moorer's methods for using probabilities for determining the use of certain intervals between adjacent notes have become an important component of more contemporary computer music generation work.

In addition to using a*ll-different* in their research into computer music composition, Michel and Hentenryck (2002) suggest the "Queens" problem illustrates a practical application of a*ll-different,* where *n* queens are placed on an *n* x *n* chessboard so no queen can attack another. This demonstrates the relatively complex solving capabilities of a simple constraint algorithm.

2.6 Music Sequence Matching

Zils and Patchet (2001) introduce the concept of "Musical Mosaicing", which aims to generate sequences of sound samples based on sample retrieval and sequence generation. This extends earlier work by Patchet on sequencing music titles, and describes how a number of constraints are applied to the problem. *Distribution constraints* control the location of different samples in the sequence according to their properties and can be used, for instance, for imposing regular rhythmic patterns. The *Pitch continuity constraint* reduces the difference between pitches of consecutive samples, and the *Title continuity constraint* aims to choose consecutive samples in the sequence, that are also consecutive in the original song from which they are extracted.

Zils and Patchet cite the work on adaptive search by Truchet and Codognet (2004) due to its suitability to solving the problem of efficient music database searching. The adaptive search is extended to include all possible samples for each segment of the "musaic". Zils and Patchet propose their system is appropriate for large scale databases, but to pre-load the constraints domain with all possible values as suggested seems entirely impractical. The process essentially builds up a random sequence based on predefined rules then browses the database for samples that better fit the required solution. Once a solution is found, the quality can be improved by *sequence refining*, which consists of "global transformations on transitions". Unfortunately there is no detailed explanation of the methods proposed for this task, so while the authors claim it will "smoothen possible rough transitions between samples", it is questionable whether this process is likely to produce valid results. While this work presents an in depth study of extracting sound samples at a low level from a database, it appears to have fallen short of the goal of generating coherent music samples. The mechanisms for music retrieval are not discussed at any point, so it is assumed the database of samples is theoretical.

Work carried out by Dowling (1971) is in the context of human music melody recognition and retrieval, but it presents some useful techniques for identifying melodies for use in automated systems. He states: "A given sequence of tones remains the same melody if each pitch is changed by the same amount". Hence it is possible to represent the melody of a song by stating interval sizes between successive tones. The intervals may be positive or negative, depending on the direction of the melody. Dowling calls this set of directional relationships between successive tones in a melody its "contour". Dowling was able to demonstrate that subjects were best able to

identify a melody by a combination of the contour plus the relative interval size. For the purposes of the research the rhythm was excluded, to allow the author to assert that recognition was possible without rhythm. For practical purposes however, the addition of at least note durations is required to extend the use of contour matching.

Uitdenbogerd and Zobel (1998) state that the task of a retrieval system is "to identify the pieces of music that contain music fragments that sound similar to the query". The work examines ways of matching pieces of music, including the sequences of notes, intervals between notes, and melody contour. The authors state that "being able to retrieve data from a music database using a melody fragment as a query would be desirable … as a composition tool". Existing work on querying by humming proved to be limited, as people do not sing accurately, but melody contour was shown to be usually accurate.

Uitdenbogerd and Zobel describe ten parameters that can be used to rank sequences:

- Exact transposition;
- Exact transposition except for a chromatically altered note;
- Exact transposition except for a diatonically altered note;
- Same contour and tonality;
- Same contour and different tonality;
- Same notes but different contour;
- Different contour and different notes.

The figures overleaf illustrate examples for each of these scenarios:

Sequence A	A	F	E	G	F#	C
Sequence B	A	F	E	G	F#	C

Figure 2.2 Exact transposition

Sequence A	A	F	E	G	F#	C
Sequence B	A	F	E	G	F	C

Figure 2.3 Exact transposition except for a chromatically altered note

Sequence A	A	F	E	G	F#	C
Sequence B	A	G	E	G	F#	C

Figure 2.4 Exact transposition except for a diatonically altered note

Sequence A	A	F	E	G	F#	C
Sequence B	B	G	F	A	C#	D

Figure 2.5 Same contour and tonality

Sequence A	A	F	E	G	F#	C
Sequence B	E	C	B	C	B	G

Figure 2.6 Same contour, different tonality

Sequence A	A	F	E	G	F#	C
Sequence B	C	A	F	F#	C	E

Figure 2.7 Same notes, different contour

Sequence A	A	F	E	G	F#	C
Sequence B	C	D	D	A	E	F

Figure 2.8 Different notes, different contour

Figure 2.2 shows exact transposition where both sequences would sound identical to the human ear. This would produce the highest possible ranking, but as the aim is to produce original music, it is likely that this sequence would be rejected. Figure 2.3 shows identical sequences except for a chromatically altered note where a single note is sharpened or flattened, so differs by at most a single semitone. This sequence would rank high in the example sequences. The sequences in Figure 2.4 are identical except for a chromatically altered note – here a single note is different by at most a single tone and would represent a similar or slightly lower ranking than the previous example sequence. Figure 2.5 shows same contour and tonality, which means the notes have the same relative pitch, but are different notes. These sequences may sound identical to the human ear, but this depends on the size of the difference and how well trained the listener is. Again sequence A essentially duplicates sequence B, so may be rejected (although it is physically different, so may avoid musical rights violation). The sequences in Figure 2.6 follow the same contour but are of different tonality. As a result, the sequences may sound very different but could successfully be played concurrently as a two-part harmony. The notes in each of the sequences in Figure 2.7 are the same, but arranged in a different order so the two sequences will sound very different and essentially share no commonality, and hence would rank low down in the list. Finally, Figure 2.8 shows a pair of completely unrelated sequences that would be ranked lowest, or be rejected.

Uitdenbogerd and Zobel describe how they stored music as a text representation of

note sequences and durations, organised into *n-grams*, which as previously discussed, are a basic sequence of *n* consecutive notes. The *n-gram* allows precise comparison of sub-sequences within larger sequences of musical data. A limitation of this approach is that a particular n-gram being searched may extend into sequences that belong to other groups of sequences within the music sequence as a whole. An extension of the *n-gram* approach is suggested which searches the sequence for the start of a clearly identifiable phrase that forms the start of each n-gram. The concept of the *flexible n-gram* is introduced, where n is truncated when a phrase boundary is encountered, such as the start or end of a repeated phrase. The basic contour mechanism that was discussed in the introduction produces a limited two-dimensional contour, with no guide to the size of change in interval. Uitdenbogerd and Zobel recommend further investigation into more precise methods, which might be more appropriate for large databases.

An important point to consider when using sequence contour matching is suggested by Downie (2000) - the absence of note duration and other song dynamics means that it becomes very difficult to distinguish one song from another. Downie suggests the use of ranked retrieval methods to compensate for loss of retrieval. It would seem simpler to not remove that additional information in the first place and to extend the searching algorithms to account for the additional items, perhaps with a second pass. The algorithmic summary provided by Downie describes a complicated algorithm for converting intervals and contours into characters depending on the direction of the contour.

A vectored sequence contour system is described by Kim (2000), which uses a five-level contour method to represent a much higher level of detail of structure and rhythm than the direction-only method described by Uitdenbogerd and Zobel, and appears to provide a solution for dealing with the limitations suggested by Downie.

-03	**+05**	**+01**	**+01**	**+01**

Figure 2.9 Vectored sequence contour

Figure 2.9 shows a vectored sequence contour based on a single level of the five-level contour suggested by Kim. There are many additional parameter values represented in this method, which are required by the MIDI standard, but are not required to facilitate contour sequence matching, so are not shown in this example.

Downie suggests that the length of the n-gram affected performance; the larger the n-gram the longer the execution time, and an increase in the occurrence of query errors were observed. However he does not clearly explain the concept of the query error, but it is assumed that a long n-gram may find no matches at all, resulting in an error condition in matching. Downie implied that a length of six provided the best performance, while a length of four produced more accurate results. It seems highly unlikely that an n-gram with a fixed length of four could produce much more than an approximation, but it appears that the time signature and default note length were not included in this assessment. The work by Downie suggested that the location of the search in the database of music sequences had no effect on retrieval effectiveness, which suggests that information useful for retrieval is distributed throughout each song. Downie does not specify the criteria for defining effectiveness, so it is difficult to tell whether this means the performance was constant, or simply that similar contours were identified regardless of which part of the music database was searched.

2.7 Music Databases

Much of the work on storing music data in databases follows the trend of persisting pure binary music data in order to facilitate searching and retrieval using methods such as humming, as used in work by Chen (1998) and Ghias (1995). While this is appropriate for accurate retrieval and reproduction of the original source of music, contour matching at a general level is difficult if not impossible. Downie (2000) makes use of a symbolic, text-based representation of music, based on the MIDI standard, which allows music to be stored in a general format that can easily be searched for patterns and styles. The definition of the MIDI standard specifies absolute values for note frequency and duration.

Figure 2.10 below shows MIDI values for C in three octaves. This illustrates that to compare notes in different octaves, the MIDI values must be adjusted.

Note (Octave)	MIDI Value	Frequency (Hertz)
C_1	12	16.3515978313
C_2	24	32.7031956626
C_3	36	65.4063913251

Figure 2.10 Standard MIDI sample

The majority of works reviewed do not specify the database mechanism in any detail. However, the format of the data within the database is vital, and the literature search revealed that this largely belonged to two categories – a binary waveform or token-

based representation. Downie (2000) suggests that a token-based representation provides the greatest flexibility for storing and searching music data. However, the MIDI standard used by Downie includes a considerable amount of additional music structure data that is not required for contour matching.

2.8 Summary

From early investigations by Moorer (1972) to more contemporary research by van Hoeve (2001), Anders (2007) and Patchet and Roy (2004), much of the work on computer assisted music composition has made extensive use of constraint programming. It is clear this technique suits the requirement for producing inexact or approximate sequences, and some interesting results have been produced. A number of refinements to these techniques have been investigated, such as the concept of randomness, which is introduced by Zimmerman (2004). Work by Truchet and Codognet (2004) supports the techniques suggested by Zimmerman and clearly illustrate the suitability of constraint satisfaction for computer music composition. Michel and Hentenryck (2002) demonstrate the adaptability of constraint programming, who propose the "Queens" problem as a solution for music composition.

The use of contour matching has been used in sequence matching for some time, and has yielded positive practical results (such as in human genome sequence research). Zils and Patchet (2001) use contour matching and a music database to produce interesting results with a concept they call "musical musaicing".

Although the research carried out by Dowling (1971) belongs within the context of music retrieval, he provides some useful concepts in contour matching that are useful for this research and its study into comparing and measuring music sequences. Uitdenbogerd and Zobel (1998) suggest the task of a retrieval system is "to identify the pieces of music that contain music fragments that sound similar to the query", which also provides some useful fundamentals for this research.
Limitations have been discovered in work on contour matching; Downie (2000) identifies that the absence of note durations and other musical dynamics in the contour matching process results in weakness in music identification (although music *pattern* identification is the primary objective in the context of this research project).

There is a considerable range of sequence contour methods in use, from query by humming (Chen, 1998 and Ghias, 1995), to a complex five-level method proposed by Kim (2000). These methods appear too complex for the needs of this research, however they do offer some useful concepts. An alternative method employed by Downie (2000) uses text based tokens for music representation, which is ideal for identifying patterns in music, rather than specific music phrases.

There appears however to be little evidence of combining constraint-programming methods with contour matching techniques, which leads to the prospect of some aspects of this research treading new ground.

3 RESEARCH METHODS

3.1 Introduction

The chosen research method consists of a number of controlled experiments involving an experimental software prototype. In order to effectively design the prototype, the primary subject area is divided into the following sub-categories:

- Music sequence generation by constraint programming;
- Music database structure;
- Sequence contour matching mechanisms;
- Contour-to-constraint feedback processing.

There are a number of challenges presented by these considerations:

- Which constraint algorithms are most suited to the problem area;
- How to store music in a format that supports sequence matching;
- How to break up whole musical pieces into fragments that can be analysed effectively;
- How to collect the results of matching so they can be fed back into the constraint composer program;
- How to modify the constraint variables to effect changes in the resulting compositions.

3.2 Music sequence generation by Constraint Programming

The use of constraints for computer-assisted music composition is divided into two

categories – general constraints that have been proven by research and experimentation, and custom constraints that are written specifically to solve the problems involved in original music composition. The use of custom constraints is limited, largely due to the relative success gained from the use of general constraints. Throughout the review of the current body of work, it is apparent that two general constraints are popular within the research subject area; *All-different* and *All-interval,* which are both sequencing constraints. All-different is the most simple, and states that each integer in a set must be pair-wise different. All-interval states that, not only should the sequence of integers generated be unique, they must contain every interval in the series.

Michel and Hentenryck (2002), Patchet and Roy (2004), Zils and Patchet (2001) and van Hoeve (2001) all suggest the use of all-different, whereas Anders (2007) advocates the use of all-interval. There is evidence that both these constraints have provided valid solutions, but we are reminded by Truchet and Codognet (2004) that given the same fundamental properties, the CSP provides predictable solutions, a limitation that applies to both these constraints. To counteract this limitation, a more complex implementation of all-different is described by Michel and Hentenryck in the CSP definition of the n-Queens puzzle. This combines three all-different constraints in order to provide a *combinatorial substructure*, which is ideal for this research as it provides an almost infinite set of solutions. The Queens CSP specifies three constraints; the first states that the chess queens must be placed on separate rows on the board, and the second and third state that the queens must be placed on different diagonals. In the context of this research, the first constraint was used to produce a sequence of unique intervals on separate lines on the stave, and the second and third

constraints were used to prevent any strict ordering of sequences from appearing in the solutions.

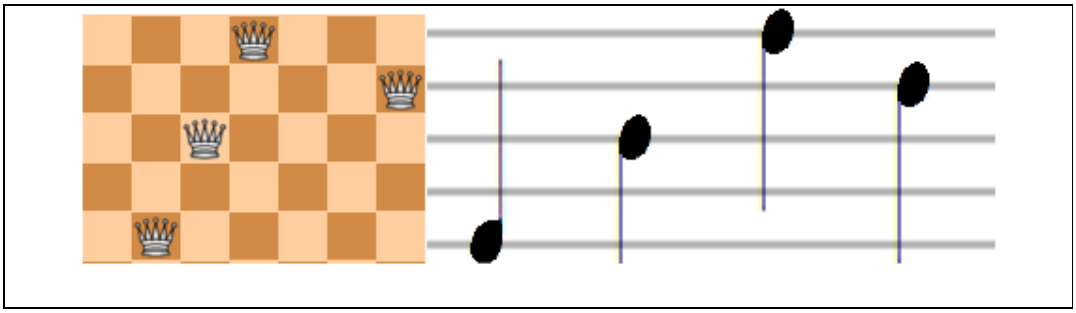

Figure 3.1 n-Queens CSP used for music sequencing

Figure 3.1 shows how the CSP for the n-Queens puzzle was adapted for use as the music sequence generator. In this example, the first constraint alone might produce a solution with three rising notes as shown, but these may be followed with three falling notes of the same interval, resulting in a sequence resembling practice scales.

3.3 The Music Database

The music database provides the reference data for the prototype composition tool, and must contain as large a sample as possible of human composed music. There are a number of sources of human music data, from published music scores to media containing actual recordings. There are a number of Internet sites that host downloadable collections of music, often in a standard format such as ABC notation. ABC notation (Walshaw, 2006) is a text-based music notation that can be read directly by a musician, and is supported by tools such as ABC Navigator[2] that can

[2] ABC Navigator is created by J P Martin and is available from http://abcnavigator.free.fr

play ABC music through the computer, and convert the music into standard music notation scores.

```
X:504
T:Hanged Man's Reel

M:4/4

L:1/8

K:G

A | B3 B BcBA | GABG AGBG | AGAB cBAG |

    AB-BB- BcBA | GABG AGBG | AFDF G3 :|

E | DGGF GFGB | AFAc B2 B2 | DGGF G2 GB | AFDA G3 :|]
```

Figure 3.2 Sample ABC notation

Figure 3.2 shows a simple tune in ABC notation. The header includes an identifier (X), the title (T), the meter (M), the default note length (L) and the key (K). The body shows notes in character form (A – G), along with optional duration information (B2) and other structural music data, such as the bar line (|) and the repeat (:|) symbol.

The specific structure of the database is of little consequence to the contour matching mechanism, as long as the music data can be retrieved from the database in a format suitable for contour matching.

The method finally selected for providing the required database data format was based on the ABC notation, resulting in minimum conversion of data when storing and retrieving music fragments.

3.4 Contour Matching

There is a common theme in music sequence matching research; the music is broken down into fragments of a specific length and the fragment, known as an *n-gram*, is converted into a contour that represents the change in interval direction and in some cases, magnitude. Downie (2000) claims the size of the n-gram is critical; too small and the frequency of matches is too high to allow effective analysis, too large and contours never match. A method for sequence contour has been selected that sits somewhere between the direction-only technique suggested by Uitenbogerd and Zobel (1998) and the five-level contour method used by Kim (2000).

Sample contour		+01	+02	+03	-01	-01							
Database tune	C	D	E	G	F#	F	|	D	E	C	E	E	B
Tune contour		+01	+02	+03	-01	-01			+01	-04	+04	+00	-05
Match score						5							1

Figure 3.3 Contour method

Figure 3.3 shows how a contour is matched against human composed music in the database. Note the "|" symbol denotes a bar end and that the contour vector value represents the interval size in semitones.

In this example, there is an exact contour match between the contour "`+01+02+03-01-01`" and the first bar in the tune, and a partial match with the second bar. This produces a "score" of six. In a large music database, contour scores could potentially range from zero to the order of thousands.

In the majority of works examined in the literature search, the objective of music contour matching was to identify and retrieve sequences with a particular contour. I

have adapted this approach to count the number of matching contours in the music database.

3.5 Contour to Constraint Feedback

An exhaustive literature search was unable to identify any work that uses automatic feedback from contour matching systems into constraint-based music composition programs, in order to improve the structure of generated sequences.

The objective of this technique is to attempt to minimise the production of solutions that are incompatible with music structures in the database, and to maximise those with similar structures and so produce music fragments of higher quality.

The score that is produced by the contour matching mechanism is fed back to the constraint satisfaction system in order to adjust the constraint program variables. The details of this technique are described in the following section on system implementation.

3.6 Whole Tune Assembly

Experiments were designed to study the feasibility of assembling sequences created by earlier experiments into "complete" tunes. Sequences were to be assembled into "phrases", then into "parts", and finally into whole tunes. This technique utilised the musical structure of a "reel" as a template.

A reel is a type of traditional dance music that is normally composed in 4/4 time, and often consists of two parts; A and B. The two parts are constructed from phrases of two bars, grouped into four and then eight. The two parts represent a "question" and "answer" scheme (Wikipedia$_3$, 2007)

Part A				Part B			
Phrase 1		Phrase 2		Phrase 3		Phrase 4	
Bar 1	Bar 2	Bar 3	Bar 4	Bar 5	Bar 6	Bar 7	Bar 8

Figure 3.4 Structure of a reel

Figure 3.4 shows the structure of a reel, which is built up from phrases containing two bars. In a reel, parts A and B are repeated in the format AABB, hence a total of 32 bars are played.

An automated method was designed based on SQL queries to assemble whole tunes using the structure scheme above. The highest scoring sequences would provide the basis for creating the phrases, whilst ensuring the intervals between the bars were suitably close.

A simple experiment was designed which has the objective of building an AABB structure tune purely by listening to sequences, and subsequently assembling them according to which combinations sound best.

4 IMPLEMENTATION

A number of limitations were applied to the practical aspects of the research to ensure effective experiments could be carried out within the available timescales.

Scope for experiments in generating music sequences were constrained by the following:

- Music range is limited to two octaves;
- All sequences are diatonic;
- All sequences are simple monophonic melodies.

This section describes how each component of the experimental tool was implemented. The tool was written in the standard C++ language and contained the constraint programming libraries and an interface to the database system. A simple user interface was provided that allowed three controlling parameters to be supplied by the user.

User Command	Description
Music.exe C 4 3	Generate 4 bars in the key of C major with a maximum interval of 3 semitones
Music.exe Bm 8 4	Generate 8 bars in the key of B minor with a maximum interval of 4 semitones

Figure 4.1 Prototype tool user interface

Figure 4.1 shows the user interface of the prototype tool and the available user options. The first parameter is the key signature, and is provided in standard music notation. The first letter of this parameter represents the keynote, and an optional

following “m” denotes a minor key. The second option is the number of bars that are to be generated, based on a time signature of 4/4. For the purposes of this research, the time signature was fixed at 4/4 and not alterable from the user interface. The final parameter is the maximum interval allowed between notes, including the maximum interval between the last note of each bar and the first note of each subsequent bar. The sequences generated by the tool are saved in the music database, and also written out to the file system as small ABC notation files in the format “tune*nnnnn*.abc” where *nnnnn* represents a unique tune number. If ABC Navigator is installed on the computer on which the music is composed, double clicking on the output files automatically loads the tune and displays it in standard notation. The tune can also be played on the computer’s sound system directly from within ABC Navigator.

4.1 Constraint Satisfaction System

I used Gecode constraint system version 1.3 (Schulte, 2007) to provide the necessary constraint algorithms. The Gecode generic constraint development environment is implemented as a standard C++ library, which was built into the prototype tool. The constraint system was supported by a number of control mechanisms, which allowed contour matching scores to be retrieved from the matching system in the database, and fed back into the constraint program. The pseudo-code fragment below illustrates the flow of execution within the tool during the composition process.

```
User enters parameters
Retrieve keynotes from database
Set branch enumerator option to default
Repeat
        Repeat
                Execute constraint search
                Match contour with music database
                If score is below threshold
                        Change branch enumerator option
        Until contour score is greater than score threshold
        Store generated sequence in database
Until required number of bars has been generated
```

Figure 4.2 Prototype tool execution flow

Figure 4.2 illustrates the iterative nature of the prototype program. The action of the user entering the parameters starts the process, and execution continues until the required number of bars of music has been composed.

Apart from the definition of the appropriate constraint algorithm, Gecode provides a number of options for controlling the search process; number of iterations, search space size, and variable selection.

The number of iterations determines how many times the search can attempt to produce a set of solutions and in theory can be any number between one and infinity. The optimum value used in this research is discussed in the results section.

The search space size determines how many proposed solutions the constraint system can generate before it can begin the branching and evaluation process. The size must be large enough to provide space for a realistic number of solutions, but not so large as to cause impractical branching execution times. A value that provided satisfactory results was obtained by experimentation and this is discussed in the results section.
There are two branching variable selection parameters that can be modified – the variable over which to branch, and which values to select first.
The following options for selecting the branching variable were used:

- With smallest min;
- With smallest domain size;
- With smallest degree;
- With largest degree.

The degree of a variable is defined as the number of dependant propagators, so a larger degree will result in more extensive branching.

The following options for selecting which values to select first were used:

- Select smallest value;
- Select median value;
- Select maximal value;
- Select lower half of domain;
- Select upper half of domain.

This facility for controlling the solutions appeared most likely to provide useful variations. Results of experimentation with the branch selection variables are discussed in section five.

The solutions produced by the CSP consist of sequences of integers that are used to represent notes.

Solution values	17	15	24	25	30	18	32	34	37	36	27	30
Corresponding notes *octave*	E_1	D_1	E_1	C_2	F_2	F_1	G_2	A_2	C_3	B_2	D_2	F_2

Figure 4.3 CSP sample solution

Figure 4.3 shows a sample solution in the form of a sequence of integers, and the corresponding notes they represent. Because the *all-different* constraint produces all the values between 1 and the maximum size of the search space, the solution may contain values that do not represent valid notes. These non-consistent values were used to derive the note durations, and hence rhythm.

The default note length used throughout the research was a crotchet or quarter note, as this is the foundation note length for most simple tunes. Weighting factors were applied to off-key notes to produce the note duration patterns. If a note in the collection of values in any given solution was determined to be off-key (i.e. not part of the chosen key), it was used to modify the duration of the subsequent note, based on the weighting factors shown overleaf:

Duration	Factor
Half note (minim)	5%
Quarter note (crotchet)	85%
Eighth note (quaver)	10%

Figure 4.4 Note duration weighting

The weightings shown in Figure 4.4 were derived from observations of the structural patterns in the database of human composed music.

4.2 Music Database

Contour matching is a process that is heavily data driven, and requires sequence data to be stored in a consistent format that supports fast searching and retrieval. Rather than attempt to develop a system to provide this functionality, a standard relational database management system (RDBMS) was used. Microsoft ® SQL Server Express© follows ANSI[3] SQL standards, and is free for students.

[3] American National Standards Institute

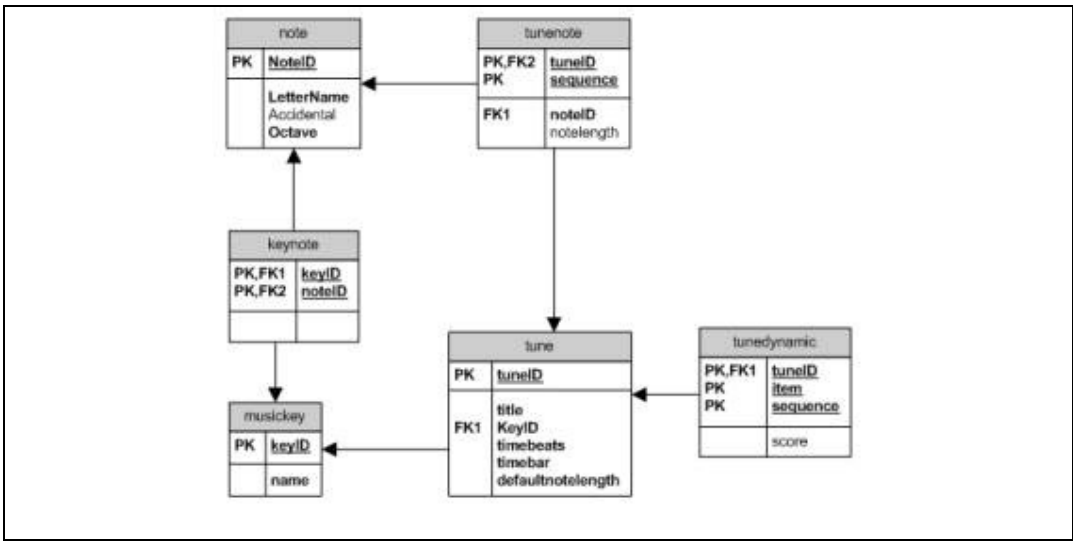

Figure 4.5 Database model

Figure 4.5 shows the database model, designed in accordance with entity relational modelling described in The Open University (2000) documentation.

A simple table in the database contained all the available notes required for the experiments. This is shown in Figure 4.6 below:

note	
noteid	INTEGER
note	CHAR(1)
accidental	CHAR(1) NULL
octave	INTEGER

Figure 4.6 The "note" table

For the purposes of this research, the note table contained notes ranging four octaves from C_0 to C_4, although the database was designed to support the whole range of octaves. The notes were loaded into the database using simple SQL insert statements as shown in Figure 4.7 below:

```
INSERT INTO note VALUES(1,'C',null,0)
INSERT INTO note VALUES(2,'C','+',0)
INSERT INTO note VALUES(3,'D',null,0)
INSERT INTO note VALUES(4,'E','-',0)
```

Figure 4.7 Note data creation script

Note 1 is C, in octave 0, with no accidental, whereas note 2 represents "C#" as it is sharpened by the '+' token.

The *keynote* table contains the collection of notes belonging to each key scale for the four-octave range selected for this research.

Keynote	
keyID	INTEGER
keyname	CHAR(100)
noteID	CHAR(1)

Figure 4.8 The "keynote" table

The *keynote* table illustrated in Figure 4.8 links to the *note* table to ensure data consistency is maintained, i.e. the each notes in each key must exist in the note table.

An important facility provided by *SQL Server* is that of database "stored procedures" and "functions", where pre-defined SQL queries are stored in the database and can be executed repeatedly. The procedures developed for this project are illustrated overleaf:

Procedure name	Parameters	Description
GetContourFromSequence	Integer sequence	Creates the contour string from the supplied list of integers
GetKeyNotes	Key letter	Retrieves the set of notes in the selected key
RecreateABC	Tune identifier	Returns a string containing the ABC notation text for the selected tune
SearchContour	Contour string	Searches the whole database for the supplied contour string and returns the count of occurrences

Figure 4.9 Database procedures

Figure 4.9 shows the main database procedures that provided the necessary functionality for sequence contour matching and for creating ABC representations of the composed music sequences.

The human composed data discussed in section three was imported into the database using a simple C++ program called ABC2DB (See Appendix B for program listing). The tool converted ABC notation data into standard SQL insert statements that were subsequently executed within the relational database system, resulting in the appropriate tables being loaded with data, as shown overleaf.

Sample ABC Tune	X:1272 T:Trip to Cartmell *M:4/4* K:D ceeg aeeg | fddf gfed | [Bd]e ef/g/ aeeg | fd f/g/a/f/ e2d2 :: cAAc BGGB | cAAg aeed | cAAc BGGB | cAAB c/d/d B2 :|
ABC2DB Output	INSERT INTO song VALUES(1272,'Trip to Cartmell','D',4,4,8) INSERT INTO songnote VALUES(1272, GetNoteID('c'), 1) INSERT INTO songnote VALUES(1272, GetNoteID('e'), 2) INSERT INTO songnote VALUES(1272, GetNoteID('e'), 3) INSERT INTO songnote VALUES(1272, GetNoteID('g'), 4) . . .

Figure 4.10 ABC notation conversion to SQL

The example in Figure 4.10 shows how a tune in ABC format is converted into data that can be stored in a standard relational database. GetNoteID is a database function that converts the note into an integer value. This function accepts ABC notation

characters to determine the octave (i.e. upper and lower case letters) and accidentals (i.e. "^", "=" and "_"). The second parameter represents the sequence of the note in the tune.

The database was populated from a library of traditional tunes in ABC format called RRTunes, created by Richard Robinson of Leeds University[4]. This collection contains more than one thousand jigs, reels, marches and other traditional folk tunes.

4.3 Sequence Contour Matching

The database procedure SearchContour illustrated in Figure 4.9 is called with a sequence of integers as the parameter input values. This sequence represents a consistent solution from the constraint satisfaction system.

CSP Solution	11,13,17,15,19,21,18
Contour	+02+04-02+04+02-03

Figure 4.11 Sample CSP solution and contour

Figure 4.11 shows a sample solution sequence, and the corresponding contour string that is produced by the *SearchContour* procedure before searching the database for occurrences of this contour. The search starts with the first tune in the database and extracts a contour of the same length as the constraint-generated contour from the first bar in that tune. The two contours are matched and the degree of matching is recorded as a numeric value within the range {0, 1, 2,…,*L*-1}, where *L* represents the length of the composed sequence. Therefore, if two ten-note contours matched exactly, a maximum score of nine is achieved.

[4] http://www.leeds.ac.uk/music/Info/RRTuneBk/gettune/00000d7c.html

This process is repeated for each bar in the tune until the contour has been compared to the whole tune, then the next tune in the database is located and the matching process continues as above. On completion, an overall matching score is produced, which is the sum of the matching scores for every sequence in the database.

4.4 Contour Feedback to Constraint System

The contour feedback mechanism is the key component of the practical research, as it provides the mechanism for comparing composed sequences and human composed music, and produces quantitative contour matching values for analysis.
The score produced by the contour matching process described in section 4.3 is obtained from database routines, and the following actions are taken, depending on the score value. Figure 4.12 overleaf illustrates the conditions used to determine what action is taken, depending on the score produced by the contour matching process.

Solutions from the initial experiments were matched with the database and the contour scores were collected and the generated sequences were saved, regardless of the score. This provided a set of control scores showing the raw results from the constraint program without any feedback. The distribution of scores provided by the initial investigations allowed a threshold value to be determined, and sequences that scored lower than this threshold were rejected.

Condition	Action
Contour score is below the minimum score threshold	The sequence is rejected and the constraint selection option is reset to the default
Contour score is above the minimum score threshold but lower than the score obtained from the previous sequence	Alter constraint selection option and re-generate the sequence
Contour score increases	The sequence is saved in the database and the constraint selection option is reset to the default

Figure 4.12 Contour feedback conditions

4.5 Whole Tune Construction

The automated method executed an SQL query to select the highest scoring sequences from the database and assemble them into the structure of the reel.

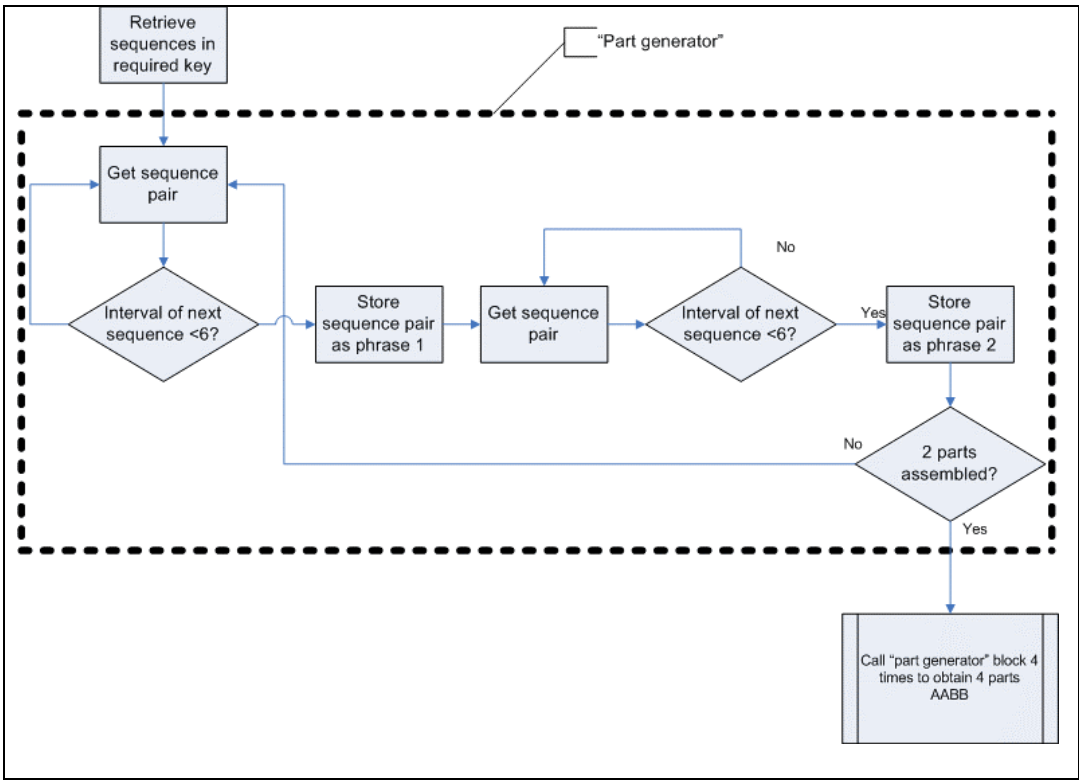

Figure 4.13 Reel construction process

Figure 4.13 shows the process flow that occurs during whole tune creation. A basic "part generator" creates "parts", which consist of pairs of phrases that in turn consist of pairs of bars.

The manual approach was much more straightforward to implement; each fragment was listened to, and the sequences that sounded best were assembled by trial and error until a satisfactory result was obtained.

4.6 Primary Results

As the primary objective of the research was to analyse music data, the prototype tool provided the primary data source for the research. The music sequences produced

were analysed using quantitative and qualitative methods, and supported by the secondary data produced by the literature search.

All composed sequences were saved in the database along with their corresponding contour matching score. During the analysis phase, the stored music data was retrieved from the database using standard SQL queries, and collated by Microsoft® Excel™. This spreadsheet software provided the statistical functions and graphical representation tools required to analyse the data and produce appropriate illustrations. Qualitative analysis entailed listening to the sequences on the computer's sound system and subsequent discussions on the structure and quality of the music fragments.

5 RESULTS

5.1 Preliminary Results

A number of initial experiments were carried out to determine the optimum Gecode runtime options. As discussed in previous sections, the available parameters are:

- Number of iterations;
- Search space size;
- Branch variable selection method.

There is no value for the default number of iterations suggested in the Gecode documentation, and the values used in the sample programs provided with Gecode ranged from 1 to 2000 so experimentation was required to determine the optimum value:

47, 53, 42, 68, 41, 46, 17, 10, 71, 67

Figure 5.1 Raw sequence with 10 iterations

47, 53, 42, 68, 41, 46, 17, 10, 71, 67

Figure 5.2 Raw sequence with 100 iterations

47, 53, 42, 68, 41, 46, 17, 10, 71, 67

Figure 5.3 Raw sequence with 1000 iterations

Figure 5.1 to 5.3 above show the raw sequences generated with 10, 100 and 1000 iterations, but with all other constraint parameters constant. Based on these results and

the values used in the Gecode examples, a nominal value of 100 was used for all subsequent experiments.

In the context of the finite integer CSP, increasing the search space size produces solutions containing more integers. However, because we are only interested in notes that belong to a given key, the increased search space produces solutions containing more invalid notes. Because of this specific requirement, exceeding a search space size of 100 produced no additional useful values, so 100 was selected as the default search space size value for all subsequent experiments.

Restricting the branching selection method was found to be a useful way of producing structural variation in the integers in each solution. It was found by experimentation that it was possible to increase the number of solutions with smaller intervals by restricting branch selection to the upper or lower half of the search space.

Optimal results were obtained by using the smallest minimum branching variable, and by using each of the five value selection options (smallest value, median value, maximal value, lower half of domain, upper half of domain) in turn until the required solution was produced.

It was found that certain combinations of branching variable selection caused Gecode to enter a “locked” state, where it never returns from branching. A new version of Gecode has been released which has a reduced number of branch selection options; one can assume that this is to avoid the above locking situation.

The literature search confirmed that contour matching requires the creation of two n-grams; the first is the n-gram representing the sequence that has been composed, and the second represents fragments of music in the database. There appeared to be two alternative methods for creating an n-gram that were appropriate, but some experimentation was required in order to determine which was the most appropriate.

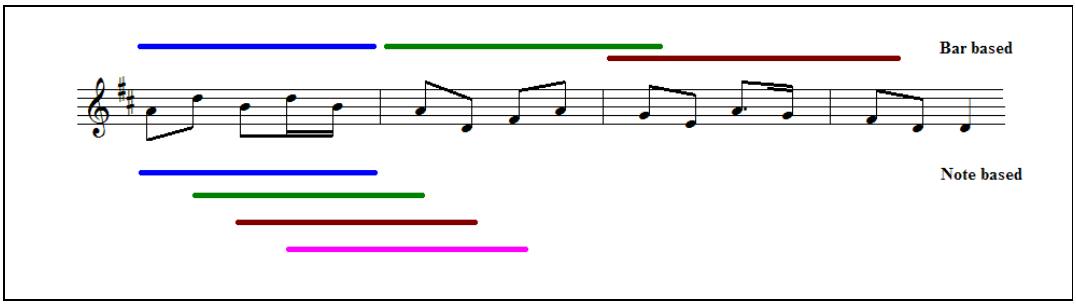

Figure 5.4 Bar and note aligned n-grams

Figure 5.4 shows a comparison between bar aligned and note aligned n-grams. Each n-gram is highlighted by a different coloured bar. A bar-aligned n-gram starts with the first note of each bar and continues for *n* notes (five in this example). A note aligned n-gram starts with every note in the tune and continues for *n* notes. This technique ensures that the matching process evaluates every note, but proved to perform too slowly to be practical for this research.

The bar aligned approach may miss the occasional note, but performed a lot faster and proved to identify contours at a suitable level of accuracy for the purposes of this research.

The bar-aligned approach may suffer from some of the inaccuracies suggested by Uitdenbogerd and Zobel (1998), as their objective was to accurately identify specific contours for retrieval. However, for the purposes of contour matching, relative change in intervals and pattern identification are the most important criteria.

Figure 5.5 below illustrates how using n-grams starting with the first note of each bar produces a very similar distribution to n-grams starting with every note in every bar.

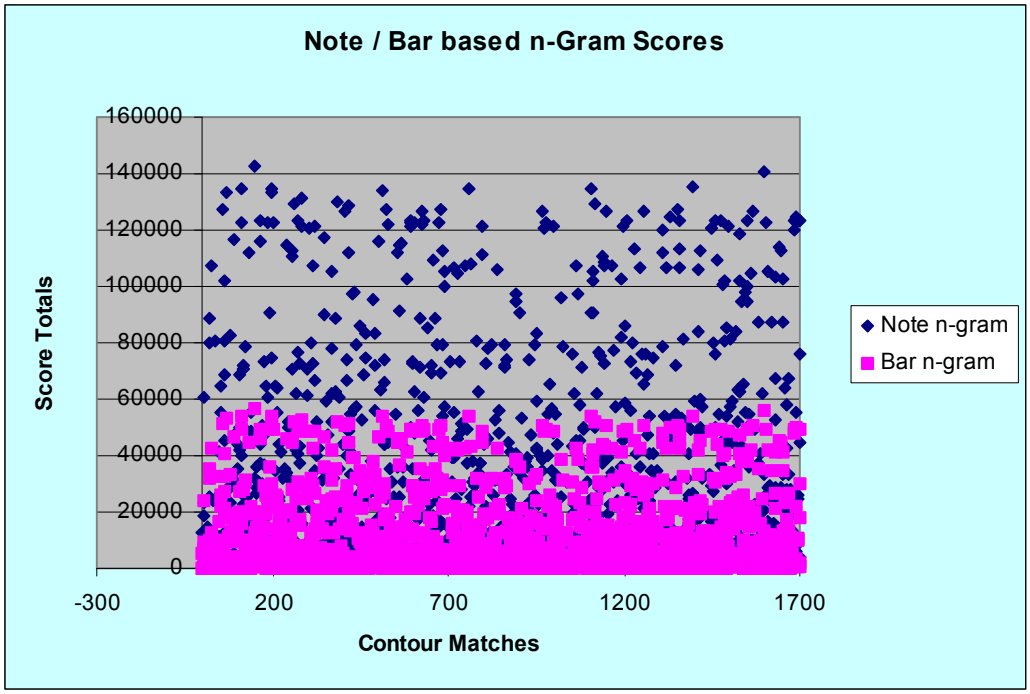

Figure 5.5 Scores with note based and bar based n-grams

5.1.1 Key Signatures

The composed samples are all shown in C major, with other key signatures represented by discrete accidentals preceding each note on the stave. The standard method for representing a key signature is a single accidental or group of accidentals immediately after the clef mark. However transposition into this representation was beyond the capabilities of the prototype.

5.1.2 Matching Performance

Experiments in contour matching directly within the downloaded ABC notation file of human composed proved rather slow. Contour-matching a single six-note sequence in the raw database took nearly 30 seconds to complete. The equivalent matching process within the relational database took just over three seconds, therefore supporting the requirement for repeatedly matching sequences throughout the sequence refinement process.

5.2 Reference music

The database of human composed music contains more than one thousand folk tunes including the sample shown below:

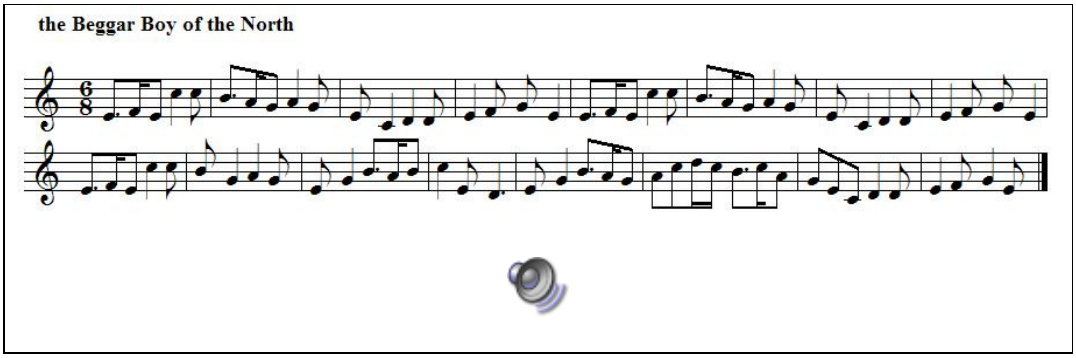

Figure 5.6 Human composed music sample

The speaker icon in *Figure 5.6* represents a sound sample that can be played directly from the electronic version of this document by double clicking the icon with the mouse. Sound samples are also available on the CDROM accompanying the paper copies of this dissertation.

5.3 Scoring

Just over 1700 sequences were generated by the contoured constraint system and stored in the database for subsequent analysis, along with the contour matching score.

The contour matching scores ranged from 0 to 56,992, and the distribution of scores is shown below:

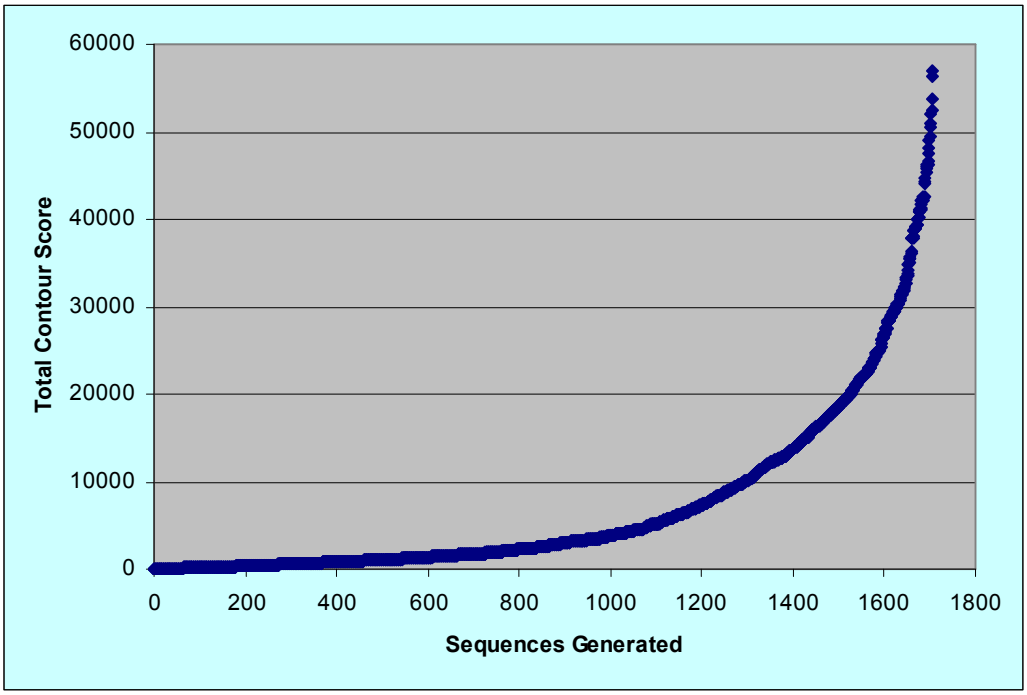

Figure 5.7 Score distribution

An almost exponential curve is shown in the score distribution shown in Figure 5.7. The sequences producing the highest scores are restricted to less than 10% of the overall number of sequences generated. I suggest this is because the difference between a "good" tune and a "bad" tune is very subtle - a sequence with a score of 100 might look very similar to one with a score of 1000, but it scores so much less

because it sounds less like the human composed music on which it is modelled. This is discussed in further detail below.

5.4 Non-Contoured Sequences

The initial group of experiments entailed generating raw sequences with no feedback from the contour matching scores. The contour matching scores were collected for analysis purposes only.

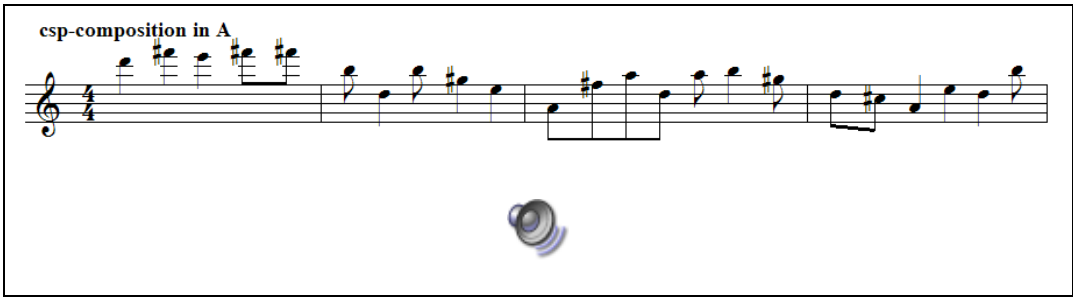

Figure 5.8 Unprocessed constraint generated sequence

Figure 5.8 shows a raw sequence that was created without applying any contour matching. This particular sequence produced a score of less than one hundred and arguably looks somewhat arbitrary and un-structured in comparison to the human composed music sample shown in Figure 5.6.

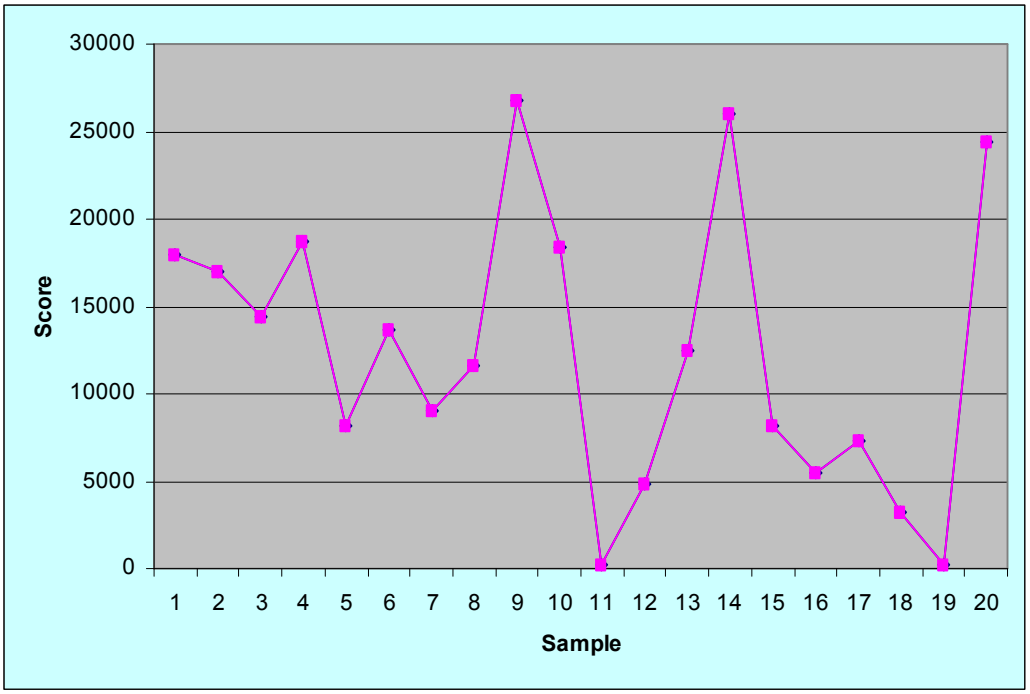

Figure 5.9 Contour scores from unprocessed sequences

Figure 5.9 shows a sample of 20 scores assigned to sequences generated without contour feedback, which illustrates that adequate contour scores were obtained from constraint system alone. This suggests that the constraint algorithm chosen to generate the music sequences meets the requirements of computer-assisted composition to some extent.

5.5 The Addition of Feedback

The process of feeding the contour scores back into the constraint generation system is now introduced, and an increase in contour scores of the resulting sequences is observed.

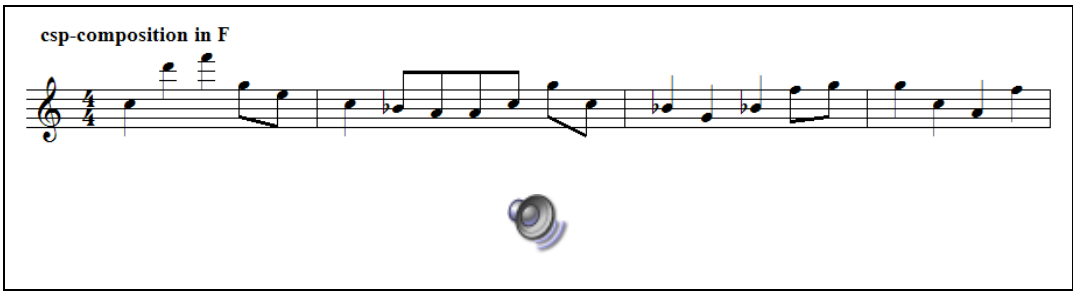

Figure 5.10 Constraint sequence with feedback

The sequence shown in Figure 5.10 produced a contour matching score of 2481. This supports the earlier suggestion that two sequences may look similar but produce very different scores, as this fragment is relatively similar in structure to the sequence in Figure 5.8. It does however *sound* better, which begins to suggest the contour feedback does in fact provide some value in improving the quality of music sequences that were composed by the computer.

5.6 Highest Scoring Sequences

The following music score illustrates an "end-product" of the contoured constraint prototype system, where contour matching has been repeated until a suitably high score is achieved and a fragment of music is, according to the results of the contour matching process, more similar in structure to that of the human composed music in the database.

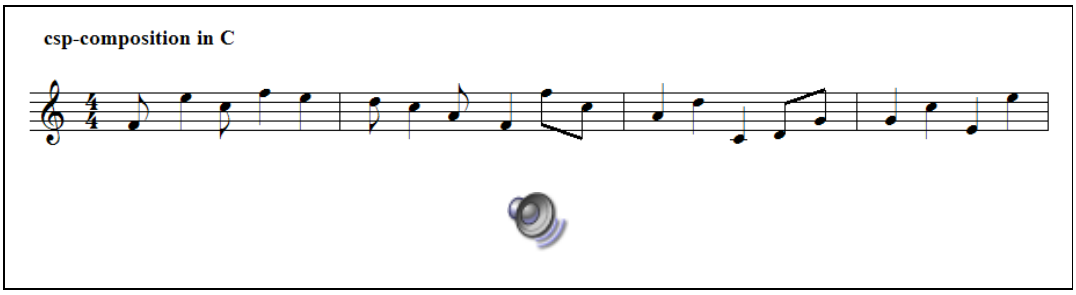

Figure 5.11 Constraint sequence with feedback

Figure 5.11 shows the score and sound sample for a sequence that scored 4818. This was the result of five attempts at "modelling" using the contour feedback score. This short tune appears to show some resemblance to the structure characteristics of the folk music in the music database.

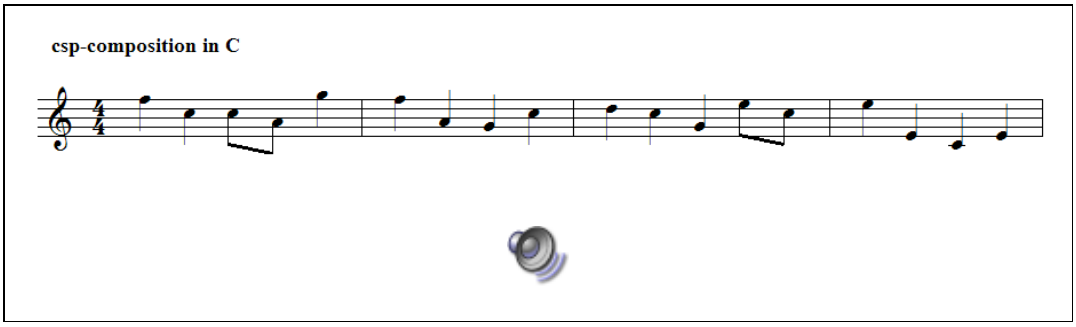

Figure 5.12 Constraint sequence with further feedback

The sequence in Figure 5.12 produced a contour score of 16081. This was achieved after 13 generations by the constraint generation system, based on scores fed back from the contour matching system.

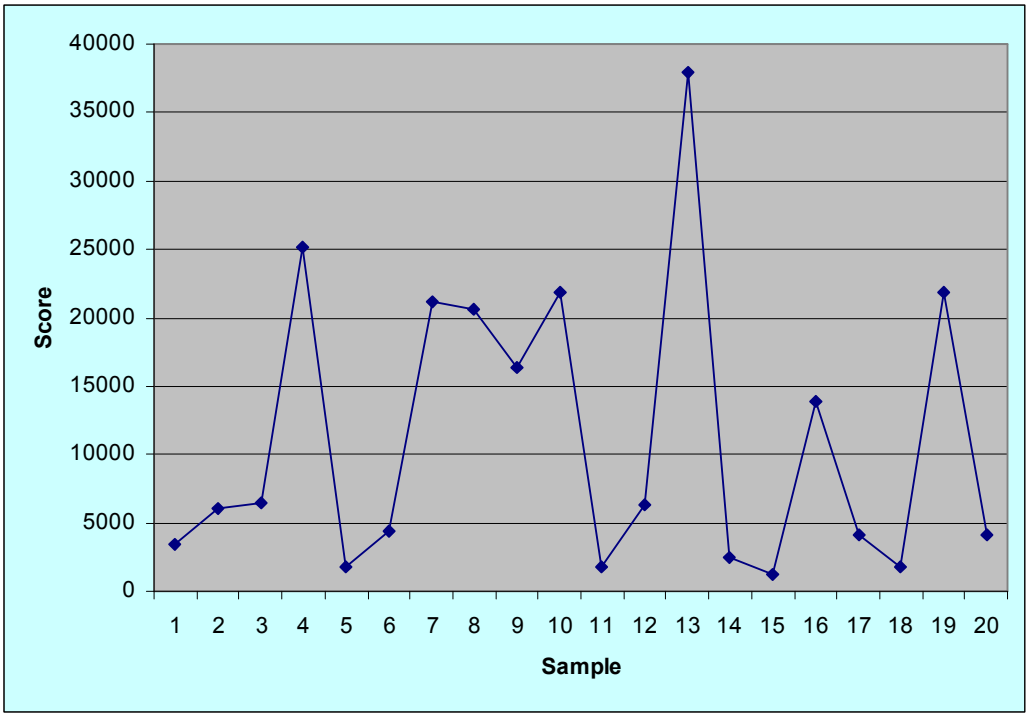

Figure 5.13 Contour scores from contour matched sequences

The sample of 20 scores shown in Figure 5.13 is taken from contour matched sequences, and exhibits a similar distribution of scores to the non-contour scores shown in Figure 5.9. The average score appears to be higher but there are many low scoring sequences, which suggests that either the contour feedback mechanism requires further refinement, or is unable to produce consistently better scores in its current form.

5.7 Statistical Analysis

A pattern is beginning to emerge in the results, which suggests higher scoring sequences contain smaller intervals. If this were the case it would corroborate the statements made by Longuet-Higgins (1993) who states that diatonic intervals (i.e.

less than six) are most common in traditional melodies. To further analyse this pattern, standard deviation is applied to samples of intervals taken from human composed music, constraint composed music, and contoured-constraint music. Standard deviation allows the variance in intervals around the mean note in each sample to be clearly illustrated. This is shown in the chart below:

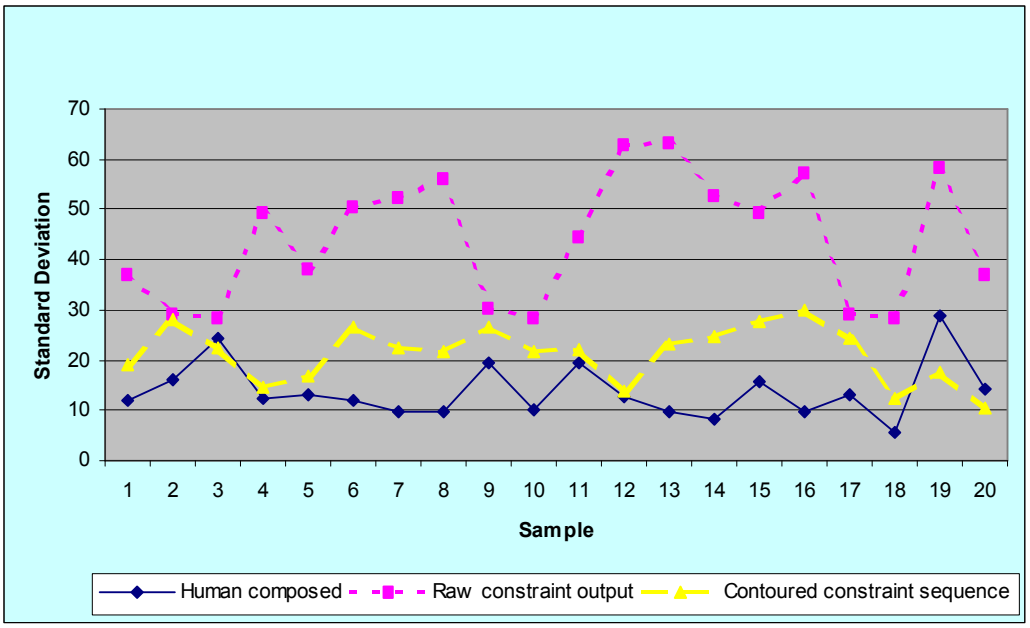

Figure 5.14 Standard deviation of music note interval

The chart in Figure 5.14 shows how the action of applying contour feedback to constraint-composed sequences results in a much closer standard deviation to that of human composed music.

5.8 Key Signatures

Six key signatures were set up in the database for the purpose of this research, although the facility to compose in different keys is data driven, so to produce sequences in a new key, the collection of notes for that key is added to the database –

no program changes are required.

Of the 1700 sequences composed, there was an arbitrary distribution of scores for each key signature, ranging from 21,303 to 30,746. This is illustrated in Figure 5.15:

Key Signature	Average Score
A	30746
Bb	27442
C	27975
D	22810
F	21303
G	26757

Figure 5.15 Average scores per key for composed sequences

There are a number of reasons why sequences composed in A might produce a much higher average than those in F. The most likely possibilities are:

- There are more tunes in the music database in A than F;
- The contours produced by the constraint program are more similar to contours in tunes in A purely by the accidental production of particular patterns;
- The three sharps in A represents a more common contour than those in F with a single flat.

Figure 5.16 overleaf confirms that there are no tunes in the database of human composed music in the key of F, and relatively few in A. This suggests that the only likely explanation for the distribution of scores is that there is subtle pattern in the solutions.

Key Signature	Tunes in Key
A	161
F	22
C	58
D	372
G	295

Figure 5.16 Count of scores per key for human composed tunes

5.9 Music Structure

The method of using off-key notes from each solution to alter the duration of the on-key notes produced some interesting structural patterns.

The sequences illustrated in Figure 5.10 and Figure 5.11 and Figure 5.12 consist largely of quarter notes, the default note length used in this research, and a small number of eighth notes.

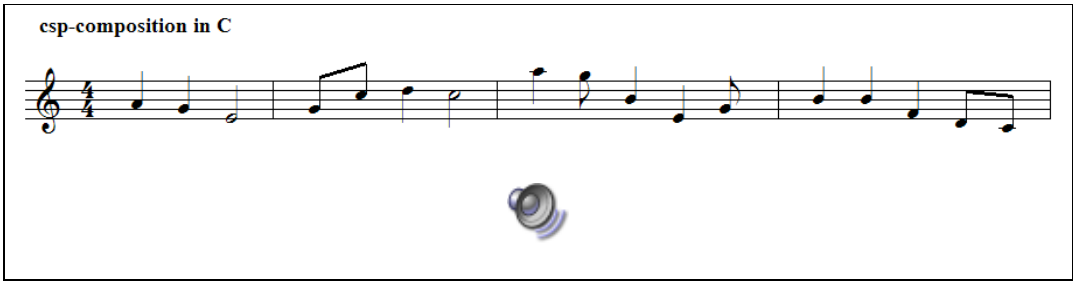

Figure 5.17 Sequence showing half-length notes

Figure 5.17 shows a composed sequence that contains half notes in addition to the quarter and eighth notes above.

Because note durations are not included in the contour matching process, it is not possible to modify or improve the rhythmic structure of the composed sequences using the method of contour matching used in this research.

The majority of solutions produced exhibited a largely arbitrary structure, where single quavers or minims occurred randomly throughout the sequence, although there was some tendency for groups of quavers to occur at the end of a bar. This suggests some pattern was being produced by the occurrence of off-key notes in the solutions. However, it was found while developing the prototype that in some cases it was necessary to override the weighting method described in section four to ensure the music adhered to music structure rules. Hence the prototype was forced to "fill up" the remaining space in some bars with shorter duration notes. This may have influenced the occurrence of groups of notes identified earlier.

5.10 Whole Tune Assembly

The constraint-based music sequence generation system was able to create repeated, but un-related sequences. The final objective was to assemble these into something resembling a complete tune.

In order to assemble the sequences into a defined structure, two experiments were carried out - the first using computer techniques alone, where an SQL query attempted

to organise pre-composed computer music sequences into AABB structure.

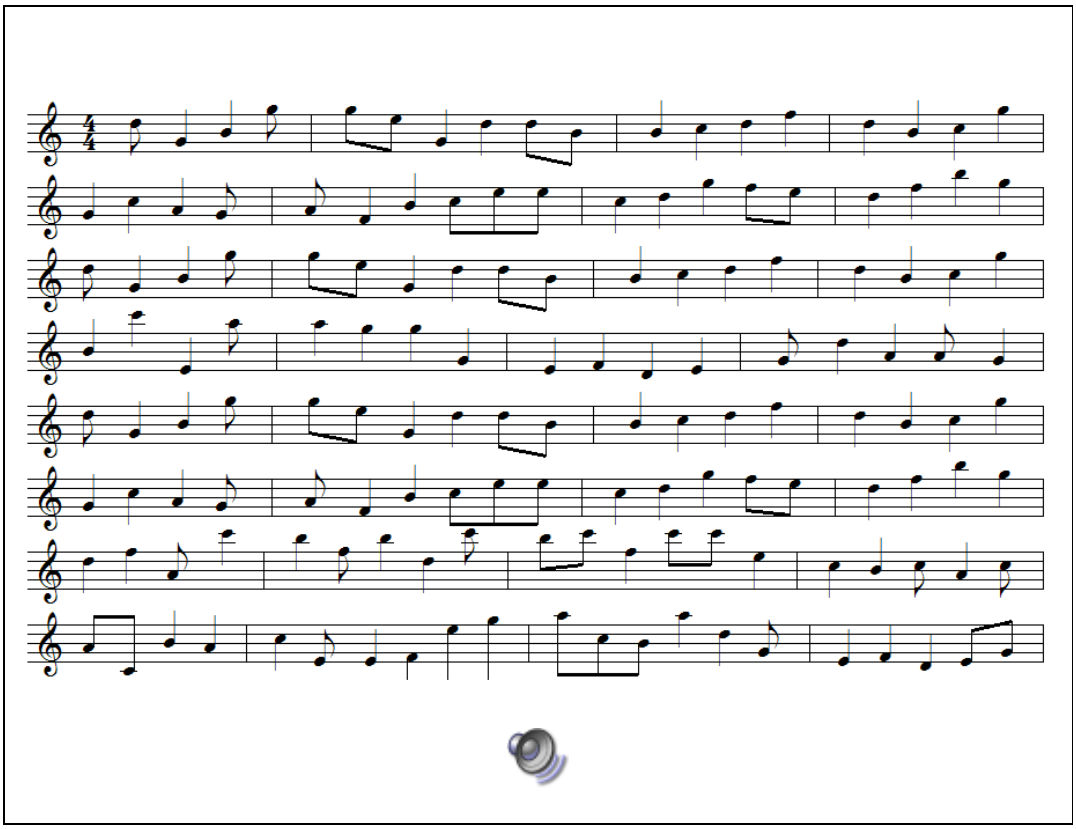

Figure 5.18 Computer assembled AABB structure tune

By visual observation of the score illustrated in Figure 5.18, and by listening to the audio sample, it is fairly obvious that the bars of music have been joined together by some mechanical process – it appears that the phrases are grouped arbitrarily, resulting in questionable musical quality.

The second experiment was a collaborative human computer approach, where computer composed sequences were arranged purely by how good they sounded together.

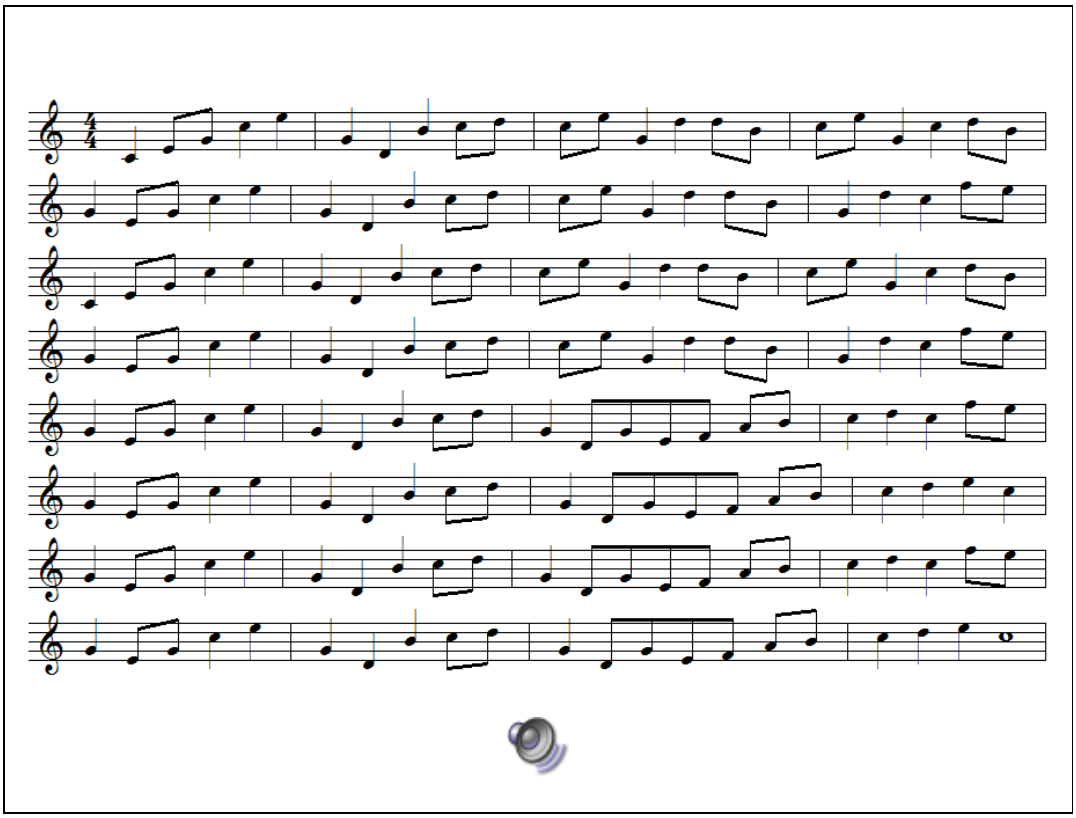

Figure 5.19 Human assembled AABB structure tune

Figure 5.19 shows an assembly of sequences that were assembled into an AABB structure by the author. The involvement of a human musical practitioner in this method introduces subjectivity into the equation.

The results from the experiments in assembling discrete sequences demonstrate that this is a complex task, and more specific techniques for structure generation are required. However, it is clear that the computer techniques involved have produced some interesting melodies that may be useful for further compositional exercises.

6 CONCLUSIONS

The objective of this research was to analyse the effects of combining constraint-based music composition techniques with contour matching against a database of human composed music, and to determine whether any improvement in quality of the computer composed music can be gained as a result of this combination of techniques.

The results obtained from the contour matching experiments corroborated the findings of Zils and Patchet (2001), Uitdenbogerd and Zobel (1998) and Downie (2000), by providing a measurable indication of how similar the computer composed music sequences were to the human composed music in the database. The process of feeding back the contour matching results into the constraint satisfaction system produced very evident changes in melodic contour. As a result of repeatedly applying contour matching feedback, the melodic contour of the resulting sequences became more similar to the contour patterns within the database of human composed music.

The techniques available for modifying the constraint programming system variables provided some level of control over the solutions provided. However, the results show that the process of re-composing fragments appeared to provide some improvement in melodic contour, but the effect of using different constraint variables on the quality of the sequences was less pronounced.

Schoenberg (1967) suggests that in order to improve composed structures further we can improve phrase endings by narrowing intervals. It was not possible to implement

this method with the constraint algorithms used in this research - to achieve this level of control, more complex constraint algorithms are required which allow multi-level solutions. As a simpler alternative, producing smaller sequences might allow the production of phrases with reducing intervals.

Some limited rhythmic structure was produced in the composed sequences, which somewhat resembled that of folk music. However, because musical structure rules were not included in the constraint definition or the constraint matching process, the extent of this element of the research was limited.

The use of a single genre of music (folk / traditional) may have had some impact on the resulting sequence structures; this type of music commonly uses repeating themes, based on bars of short notes with reasonably large intervals. The generated sequences with the highest contour matching scores contained some of these fundamental properties of folk music.

Auditory observation confirmed that higher scoring sequences did in fact sound somewhat more agreeable, but using a query to combine a selection of the highest scoring sequences into a small tune of AABB format produced disappointing results. However, it was possible to create a whole tune in AABB format possessing reasonable tonal quality by human assembly of composed sequences.

On revisiting the questions posed by the research, the results clearly corroborate the hypothesis that generated music sequences can be refined as a result of matching with a database of human composed music, and in terms of quantitative analysis, the

sequences became more similar to human music and so, arguably, are of higher quality.

The question of whether a database system provides advantages for matching is corroborated both in terms of performance and facilities for storage and retrieval of computer-generated sequences. However, the required database functionality could have been provided without the use of a proprietary system, but a lot more design and development work would have been required.

The reference data used in the music database belonged to a relatively limited genre of musical style so it was not possible to identify common patterns in different styles in this instance, although the tendency for patterns to contain groups of sequences with small intervals was corroborated.

The methods used in this research resulted in the development of a prototype that relied heavily on parameters in order to generate music sequences, so it is unlikely that this technique, as a consumer of parameters, could be complemented by learning systems to generate parameters for a given style.

6.1 Project Review

The experimental work carried out by others in the field of computer assisted composition proved invaluable for guiding the direction of this research, and the experiments carried out demonstrated some interesting effects of combining the technologies of constraint programming and contour matching.

Because of the subtleties of what constitutes a "good" piece of music, practical experimentation proved to be an ideal method for undertaking computer music research. However, the use of existing constraint algorithms proved to be a limiting factor in the way the solutions were generated, so more extensive investigations into alternative constraint satisfactions problems would have been useful.

The prototype took a considerable amount of time to design and develop due to the multi-functional aspect of the software. However, the prototype provides a useful foundation for further research and some unexpected benefits were obtained. For instance, one of the scope limitations placed on the research was to produce diatonic sequences only. Rules for controlling interval sizes evolved during the design of the prototype, and hence were available as a user-parameter and therefore are easily changed.

6.2 Future Research

There are a number of areas that represent potential for future research, based on the outcome of this project.

6.2.1 Musical Constraints

The approach of using simple pre-defined constraints for music composition resulted in some limitation in the flexibility of the composition process. The selected constraint definitions restricted the production of solutions containing repeated notes or phrases, and as suggested by Schoenberg (1967), some element of repetition is a

positive attribute for music compositions.

Because of the limited results on rhythmic structure, there is scope for designing techniques for modelling the elements of music that result in musical structure and rhythm. This includes more flexible control over note durations, the introduction of rests, and the use of accents. This research could be based on refinement or combination of existing constraint algorithms, or on the design of completely new constraint algorithms, based on detailed musical structure and composition rules.

6.2.2 Music Matching

The results gained from using basic contour matching methods provided valuable data for analysing the melodic content of the music sequences created by the constraint-based music composition program. Because of the modular design of the prototype, further experiments into contour matching using multi-genre music would require the addition of new data, and no further software development would be required. However, because of the emphasis on melody contour matching, further investigation into multi-level matching methods such as those proposed by Kim (2000) is likely to present some interesting results.

6.2.3 Music Database Re-assembly

While some of the discrete sequences in the database of computer-composed sequences were of acceptable quality, it proved difficult to assemble them into whole tunes of worthy of being called a “good” tune. Definition of methods for arranging

these into a “finished” piece of known structure requires considerable further research.

6.2.4 Polyphony

The scope of this research was limited to the methods involved in creating short, monophonic melodies with a range of two octaves. Extending the range to more than two octaves would require the addition of more note values to the database - from the prototype perspective, the rules for composition are the same.

The production of polyphonic music is more complex, and some work on using constraint programming for solving polyphonic music problems was encountered during the literature search. This is clearly an area worthy of further research, and one that is ideally suited to combining with the techniques used in this project.

References

Anders, T (2007) 'Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System', [Online] at http://strasheela.sourceforge.net/documents/TorstenAnders-PhDThesis.pdf (Accessed: 24 March 2007)

Balaban, M., Ebcioglu, K., and Laske, O. (1992) *Understanding Music with AI: Perspectives on Music Cognition*, Cambridge (USA), MIT Press, pp. 3 – 21.

Brooks, F. P. et al., *An Experiment in Musical Composition* in (Schawanauer and Levitt, 1993), pp. 471 – 481.

Chen, J. C. C. and Chen, A. L. P. (1998) 'Query by Rhythm: An Approach for Song Retrieval in Music Databases', proceedings of IEEE Intl. Workshop on Research Issues in Data Engineering, pp. 139 – 146.

Cope, D. (2001) *Virtual Music,* Cambridge (USA), MIT Press, pp. 2 – 145.

Dowling, W. J. (1971) 'Contour, Interval, and Pitch Recognition in Memory for Melodies', The Journal of the Acoustical Society of America - Feb 1971, Vol. 49, Issue 2B, pp. 524 – 531. [Online] at http://doi.acm.org/10.1121/1.1912382 (Accessed: 1 Aug 2007).

Ghias, A., Logan, H., Chamberlin, D., and Smith, B. C., (1995) 'Query by Humming: Musical Information Retrieval in an Audio Database', proceedings of Third ACM International Conference on Multimedia, pp. 231 – 236.

Moorer, J. A (1972), 'Music and computer composition', Communications of the ACM archive, Vol. 15, Issue 2, New York, ACM Press. [Online] at http://doi.acm.org/10.1145/361254.361265 (Accessed: 16 April 2007)

van Hoeve, W. (2001) 'The Alldifferent constraint: a survey', Cornell University, Department of Computer Science. [Online] at http://arxiv.org/PS_cache/cs/pdf/0105/0105015v1.pdf (Accessed: 19 January 2008)

van Hoeve, W., Pesant, G., Rousseau, L. and Sabharwal A. (2006) 'Revisiting the Sequence Constraint', Principles and Practice of Constraint Programming, Vol. 4204/2006, Berlin, Springer-Verlag. [Online] at http://doi.acm.org/10.1007/11889205_44 (Accessed: 4 August 2007)

Hofstadter, D (2001), *Virtual Music*, in (Cope, 2001), Cambridge (USA), MIT Press, pp. 36 – 42.

Kim, Y. E., Chai, W., Garcia, R., and Vercoe, B. (2000) 'Analysis of a contour-based representation for melody', URL: http://www.ragomusic.com/publications/musicIR2000.pdf (Accessed: 11 February 2008)

Kugel, P. *Computational Musicology* in (Balaban, 1992), pp. 31 – 48.

Leler, W. (1988) *Constraint Programming Languages: Their Specification and Generation*, Reading (USA), Addison-Webley, pp. 2 – 14.

Longuet-Higgins, H. C., *The Perception of Melodies* in (Schawanauer and Levitt, 1993), pp. 471 – 481.

McCready, M. and Silverman, A. (2006) 'How many hits?', Guardian Unlimited. Guardian Weekend, [Online]. Available from http://www.guardian.co.uk/weekend/story/0,,1943240,00.html (Accessed: 14 November 2007)

Michel, L and Hentenryck, P. (2002) 'A Constraint-Based Architecture for Local Search', Proceedings of the 17th ACM SIGPLAN conference, pp. 88 – 100, [Online] Available from: http://doi.acm.org/10.1145/582419.582430 (Accessed: 25 February 2007)

Patchet, F. and Roy, P. (2004) *Principles and Practice of Constraint Programming,* Volume 1713/2004, Berlin, Springer, pp. 331 - 345.

Rader, G. M. (1993) *The Perception of Melodies* in Schwanauer, S. M and Levitt, D, A. (eds.) (2003) *Machine Models of Music,* Cambridge (USA), MIT Press, pp. 471 – 481.

Schoenberg, A. (1967) *Fundamentals of Musical Composition*, London, Faber and Faber (pp. 3 – 31).

Schulte, S. (2007) 'Generic constraint development environment', KTH - Royal Institute of Technology, Stockholm, Sweden [Online] at http://www.gecode.org/gecode-doc-search-1.3.0/ (Accessed 12 June 2007)

Smith, B. (2003) 'Constraint Programming in Practice: Scheduling a Rehearsal', [Online] at: http://www.dcs.st-and.ac.uk/~apes/reports/apes-67-2003.pdf (Accessed: 14 April 2007)

Smoliar, S. (1972) 'Comments on Moorer's music and computer composition'. Communications of the ACM archive, Vol 15, Issue 11 (Nov. 1972), New York, ACM Press, pp. 1000 – 1001. [Online] at http://doi.acm.org/10.1145/355606.361896 (Accessed: 19 June 2007)

Sorensen, A. and Brown, A. (2000). 'Introducing jMusic', InterFACES: Proceedings of The Australasian Computer Music Conference, Brisbane, ACMA, pp. 68 – 76. [Online] at http://eprints.qut.edu.au/archive/00006805/01/6805.pdf (Accessed: 2 August 2007).

Truchet, C. and Codognet, P. (2004) 'Musical constraint satisfaction problems solved with adaptive search', Journal of Soft Computing, Issue 8, Vol 8, No. 9 Berlin, Heidelberg, pp. 633 – 640. [Online] at http://doi.acm.org/10.1007/s00500-004-0389-0 (Accessed: 20 May 2007)

The Open University (2000) *M876 Relational Theory,* Milton Keynes, Open University, pp. 20 – 23.

Uitdenbogerd, A. and Zobel, J. (1998) 'Manipulation of music for melody matching', Proceedings of the sixth ACM international conference on Multimedia, pp. 235 – 240. [Online] at http://doi.acm.org/10.1145/290747.290776 (Accessed 14 May 2007)

Walshaw, C. (2006) 'ABC Music Notation', [Online] at http://www.walshaw.plus.com/abc/ (Accessed 8 May 2007)

Wikipedia$_1$ (2007) 'Computer Music', [Online] at http://en.wikipedia.org/wiki/Computer_music (Accessed 2 Feb 2008)

Wikipedia$_2$ (2007) 'Steinberg Cubase', [Online] at http://en.wikipedia.org/w/index.php?title=Steinberg_Cubase&oldid=150890016 (Accessed: 16 August 2007)

Wikipedia$_3$ (2007) 'Reel (dance)', [Online] at http://en.wikipedia.org/wiki/Reel_%28dance%29 (Accessed: 4 January 2008)

Zils, A. and Pachet, F (2001) 'Musical Mosaicing'. Proceedings of DAFX 01, December 2001. [Online] at http://www.csis.ul.ie/dafx01/proceedings/papers/zils.pdf (Accessed: 22 June 2007)

Zimmerman, D. (2004) ‘Modelling Musical Structures’, Journal Constraints, Issue Volume 6, Number 1 / January 2001, Netherlands, Springer, pp. 53 – 83. [Online] at http://www.springerlink.com.libezproxy.open.ac.uk/content/g1084146148112t5/fulltext.pdf

Index

Appendix A

Glossary

Atonal	Music that lacks a key – may sound unusual or even distasteful.
Chromatic	Musical scale, comprising eleven notes, each a *semitone* apart.
Clef	Stave marker denoting whether music is played above or below middle C.
Common Music Notation	The standard written representation of music.
CSP	Constraint Satisfaction Program
Diatonic	Music using traditional major and minor scales, such as *chromatic*.
HMSL	Hierarchical Music Specification Language.
Meter	Alternative description for *time-signature* when describing a piece of music in standard music notation.
Monophonic	Music consisting of a single melody.
Octave	The interval between one musical note and another with half or double its frequency.
Polyphonic	Music consisting of more than one melody.
Semitone	Smallest interval allowed between notes in western music.
Western music	The structure of music from western civilization (not the American genre of music also called western, or country and western).

Appendix B - Prototype tool program extract

The following listing shows how the prototype was constructed and how the calls to the Gecode constraint development environment were made.

- Initialise constraint satisfaction variables:

```
char key[10];
Options opt("Music");

opt.iterations = 500;
opt.size       = 30;   // Size of search space
opt.naive              = true;
opt.quiet              = true;
strcpy(key,"D");
strcpy(opt.id,"Experiment name in ");
strcat(opt.id, key);
strcpy(opt.keyset, notesinkey);
```

- Post the variables to the constraint system:

```
const int n = q.size();
IntArgs c(n);

for (int i = n; i--; )
        c[i] = 1+(rand()%(i+1));
distinct(this, c,q, opt.icl);
for (int i = n; i--; )
        c[i] = !c[i];
distinct(this, q, opt.icl);
```

- Define the search space selection parameter:

```
branch(this, q, BVAR_SIZE_MIN, BVAL_MIN);
```

- Call the constraint satisfaction, passing the populated Options structure:

```
Example::run<Music,DFS>(opt);
```

- Contour match the generated sequence against the music database:

```
sprintf(cmd, "osql -S sqldb -E -d music -Q \"exec matchcontour '%s'\"
-o c:\\temp\\contourout.txt", opt.sequence);
system(cmd);
```

- Save the generated sequence in the database:

```
sprintf(cmd, "osql -S sqldb -E -d music -Q \"exec processsequence
‘%s’”, opt.sequence);
system(cmd);
```

Appendix C – The contour matching database routine

This is the actual contour matching routine, written as a standard Microsoft® SQL Server™ function.

```
CREATE function [dbo].[searchcontour](@sequence
varchar(1000),@maxinterval int)
returns int
as
begin
	declare @contour varchar(1000), @tuneid int, @x int, @l int, @c
int, @k int, @findcount int, @foundcontour varchar(1000)
	set @contour = dbo.GetContourFromSequence2(@sequence)
	set @l = (len(@sequence)/3)
	set @findcount = 0
	select @tuneid = min(tuneid) from tunenote
	--Iterate through all tunes in database
	while exists (select 1 from tune where tuneid = @tuneid)
	begin
		--start at first note in tune, and record last note in
tune
		select @k = 1, @c = max(sequence) from tunenote where
tuneid = @tuneid
		--Iterate through tune
		while @k < (@c - @l)
		begin
			--Get contour from position @k for length @l
			set @foundcontour = dbo.GetContour2(@tuneID, @k,
@l)
			if @foundcontour <> ''
				set @findcount = @findcount +
dbo.comparecontour(@contour, @foundcontour, @maxinterval)
			--Find next bar line
			select @k = min(sequence) from tunedynamic where
tuneid = @tuneid and sequence > @k and item = '|'
			--If no more bars, finish search of this tune
			if @k is null break
			--then next note after bar line
			set @k = @k + 1
		end
		select @tuneid = min(tuneid) from tunenote where tuneid >
@tuneid
	end
	return @findcount
end
```

Appendix D – CDROM

The attached CDROM contains the program source code of the experimental prototype, ABC notation files of composed sequences, and sound samples of composed sequences in WAV format.