A Study of Grammar-based Compression in Application to Music Analysis

a thesis presented
by
David John Humphreys
to
The Department of Computer Science & Informatics

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science

Cardiff University
Cardiff, Wales
May 2022

©2022 – David John Humphreys
all rights reserved.

Thesis advisor: Dr. Kirill Sidorov David John Humphreys

Summary

In 2014, the Computational Music Research Group of the School of Computer Science, Cardiff University, demonstrated (Sidorov, Jones, & Marshall, 2014) the utility of computing compressive context-free straight-line grammars as a powerful framework for symbolic analysis of music (with applications in structure analysis, error detection/spell-checking, and high-level editing).

The attempt to use grammar-based compression to automatically perform musicological tasks, such as pattern discovery, is a relatively new and promising technique. Using a publicly-available collection of nearly 8000 digital scores, this thesis provides an extensive investigation detailing the performance of grammars against a range of compressors, when applied to error detection, classification and segmentation tasks, and detailing a novel method of transcription error location, where grammars outperform the other algorithms tested. Grammar-based compressors are demonstrated to be a competitive tool for musicological investigation.

Increasing the coverage a single production rule can provide within the encoding of a grammar is a technique proven to be effective in increasing compression. This study demonstrates the possibility of leveraging an additional dimension in the substring search space in order to produce smaller models. Experiments confirm that even with stringent constraints the production of smaller encodings is possible, and, particularly with regard to musical data, even minor reductions are able to produce better results for analytical applications.

The study also presents a novel method of selecting substrings for inclusion in a grammar under construction, using a heuristic which loosely approximates the ability of a candidate production rule to best reduce a grammar's encoding length. Some background theory is provided as a developmental basis, after which the effectiveness of the heuristic is explored empirically, showing that a logarithmic decrease in grammar production complexity is possible, but this does not always enable the production of more compact grammars which include rule modifiers. Nonentheless, it is shown to be effective in the fast production of compact grammars.

Thesis advisor: Dr. Kirill Sidorov David John Humphreys

A Study of Grammar-based Compression in Application to Music Analysis

Abstract

Music is an important phenomenon in human civilization, about which we understand surprisingly little. In 2014, the Computational Music Research Group of the School of Computer Science, Cardiff University, demonstrated the utility of computing compressive context-free straight-line grammars as a powerful framework for symbolic analysis of music (with applications in structure analysis, error detection/spell-checking, and high-level editing).

This work explores the hypotheses that (a) constructing grammars from music data is a viable method of music analysis, with the smallest grammar producing the most significant representation, and (b) it is possible to improve upon analytical effectiveness by increasing substring similarity in order to produce smaller models.

Many studies have presented computational models of musical structure, particularly as logically-coded algorithms or deep learning models, as an important aspect of musicological analysis. However, the attempt to use grammar-based compression to automatically perform musicological tasks (such as pattern discovery) is a relatively new and already promising technique. Using a publicly-available collection of nearly 8000 digital scores, this thesis represents an extensive investigation detailing the performance of grammars against a range of compressors, when applied to error detection, classification and segmentation tasks, and detailing a novel method of transcription error location, where grammars outperform the algorithms tested. Despite making no use of specific domain knowledge, grammar-based compressors are demonstrated to be a competitive tool for musicological investigation.

Increasing the coverage a single production rule can provide within the encoding of a grammar is a technique proven to be effective in increasing compression. However, the use of domain-specific structures may provide an advantage over more general or constrained approaches to modelling equivalence. Using a novel method of increasing similarity which is fully adaptable to include general, specific or domain-based modification or production rules, this study demonstrates the possibility of leveraging an additional dimension in the substring search space in order to produce smaller models. Experiments confirm that even with stringent constraints the production of smaller encodings is possible, and, particularly with regard to musical data, even minor reductions are able to produce better results for analytical applications.

The study also presents a novel method of selecting substrings for inclusion in a grammar under construction, using a heuristic which loosely approximates the ability of a candidate production rule to best reduce a grammar's encoding. Some background theory is provided as the basis development, after which the effectiveness of the heuristic is explored empirically. A conclusion is reached, showing that a logarithmic decrease in grammar production complexity is possible, but that a strong performance boost may result in a failure to explore the search space sufficiently to produce more compact grammars overall when the modification of production rules is included. Nonetheless, it is shown to be effective in producing more compact grammars than a standard method which does not allow rule modification, and in a fraction of the time possible when using that method to explore the extended search space.

The work presented in this thesis builds on existing studies, as follows:

1. New applications of standard grammar-based compressors are investigated, including the automatic identification and reversal of data errors, classification of music into similarity-based groups, and division of the musical surface into analytically significant segments.

2. Development and investigation of a novel method of grammar encoding is conducted, which allows increased substring similarity to enable the production of smaller grammars, and aims to leverage this ability against exsting musical and non-musical applications.

3. A novel heuristic which reduces production complexity is developed and investigated, which allows complex grammars to be constructed more quickly than is currently possible with standard methods.

4. Where possible, empirical results are obtained for all methods and applications presented in this study, and these are rigorously compared to produce an evaluation of their effectiveness in each specific case.

5. A set of considerations and recommendations for future work are provided, using experimental observations to highlight promising areas of investigation by which the ideas presented in this study may be further explored and improved.

Overall, this study represents an exploration of leveraging straight-line grammars against analytical tasks, specifically given data from musical scores, and presents methods which allow the inclusion of domain knowledge to produce smaller models with strongly decreased computational complexity.

Contents

For Joe, who has patiently waited 5 years for an update on my progress.

I also dedicate this work to my late father, John – a talented jazz clarinetist – who bestowed in me two of life's greatest gifts: the appreciation of science, and the love of music.

Acknowledgments

I would like to thank Cardiff University, UK, for fully funding my PhD via stipend, without which this study would not have been possible. I gratefully acknowledge the support of the Supercomputing Wales project, which is part-funded by the European Regional Development Fund (ERDF) via Welsh Government, and whose facilities were used throughout the first set of experiments.

I would also like to thank Prof. Meredith, Prof. Marsden and the excellent reviewers at the *Journal of New Music Research* for their constructive feedback, kind correspondence and encouragement. It is through their input, and the continued dedication of my supervisors, that I have been able to grow sufficiently to make my contribution.

Finally, I would like to express my gratitude to my supervisors: Dr. K. Sidorov, Prof. A. Marshall, and Dr. A. Jones. Their unwavering enthusiasm and professionalism has been a source of constant inspiration and encouragement to me throughout my studies, and the text of this thesis is interwoven with the sharpness of the ideas collectively imagined and analysed over hundreds of high-entropy meetings, always enjoyable and unfalteringly constructive. They are, without a doubt, some of the brightest minds I will ever encounter, and it has been the greatest privilege of my life to learn from them. Perhaps most importantly, they are genuinely unique and wonderful people, and, in my opinion, incredibly talented musicians and composers. Without them, this thesis could never have existed. My heartfelt thanks remains with them all.

1 Introduction

According to Nicholas Cook, "If a few combinations of pitches, durations, timbres and dynamic values can unlock the most hidden contents of man's spiritual and emotional being, then the study of music should be the key to an understanding of man's nature" (Cook, 1994). The tantalising question of how music may be understood, and what mapping exists between composition and the often subconscious effects experienced by a listener, has been the subject of much study and philosophy, e.g. (D. Arnold, Arnold, & Scholes, 1983), (Palisca, 2000), (Clark & Rehding, 2001), (Kaser, 1993).

The analysis of structure within a musical piece is an important approach to music analysis, and is traditionally achieved through the context of music theory. Schoenberg (1967) asserted that organisation in the form of logic and coherence is what separates random noise and musical form, comparing this structure to the application of grammar to a language. Schenker (1969) was also convinced of the existence of organisation, and developed a pitch-based hierarchical model of musical form which he used to analyse a great many works. Both authors identified hierarchical elements based on pitch and rhythm, but did not offer a scientific formalism of their observations. Many other analytical approaches exist, including those which focus on performance (Lerch, Arthur, Pati, & Gururani, 2019), compositional form, and even physical gesture (Gritten & King, 2006), where a listener's physical responses may be used to map expressive dynamics to the musical surface. Structural analysis, and the role of grammars in performing such analysis, is the basis of this study.

Repetitive structures present within a musical piece may be leveraged to compress it. The Minimum Description Length principle (Rissanen, 1978) suggests that the best description of a given data sequence, such as that formed by the temporal series of notes present in a musical score, is represented by the smallest model capable of reconstructing that series with the minimum error. Since repeating patterns within note sequences may be replaced by references to a single copy of each pattern, such patterns may be used to compress a sequence; this provides one method of generating a compact model, and thus, for our purposes, measuring description length as an approximation of Kolmogorov Complexity.

Following this observation, the present thesis explores these specific hypotheses:

- Symbolic music data contains regularities which may identified using compression techniques.
- Such regularities are significant to the structure of the music, and, in turn, to the intentions of the composer with regard to the music's composition.
- Having identified such structure, the information may be used to perform useful musicological tasks, such as grouping pieces by similarity, or identifying motifs or patterns which are characteristic of a particular piece, or composer.

This work aims to show that compression may be used to solve a number of musicological tasks, despite an absence of knowledge of the musical domain.For the purposes of this work, the term "domain knowledge" may be loosely defined as the level of knowledge a person with moderate musical training may possess in this area – perhaps through academic study, experience as a musician or composer, or adherence to a particular school of analysis – which may be useful or even mandatory to their ability to perform a given musicological task. Such tasks include the detection and correction of errors, such as those which may occur during transcription; the classification of pieces by melodic characteristic; the segmentation of pieces in a manner similar to an expert human analyst; and the possibility of a structural approach to the manual editing of music. As such, it explores the following research questions:

- Can accepted string compression techniques, specifically, straight-line grammar constructors, be used to compress symbolic music data in a way which is useful to the research hypotheses, despite having no knowledge of the musical domain?
- Can grammar-based compressors:
    1. Discover and correct simplistic errors in musical sequences?
    2. Group musical works by their similarity to each other?
    3. Select note sequences (patterns) which are highly similar to those an expert musicologist would select?

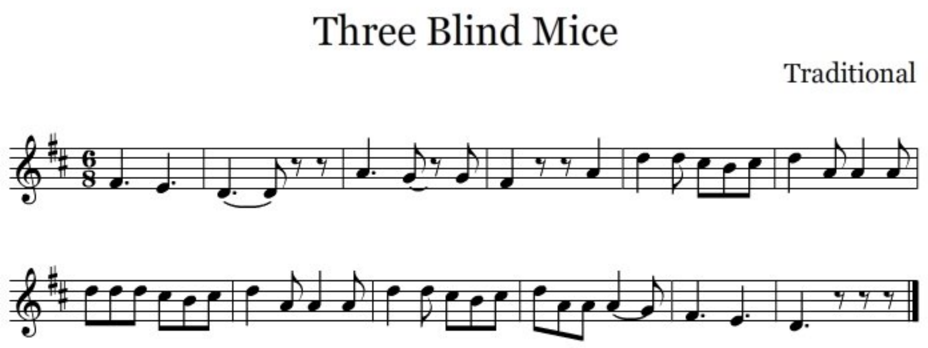


Figure 1.1: The main melody of "Three Blind Mice" (Sabrebd, 2009).

4. Compete against specialised techniques on known Music Information Retrieval tasks?

- Can a grammar-based compressor be improved – specifically, in compression ability, and its performance on the above tasks – by increasing the count of sequences which it considers *equivalent*, by widening this concept to that of *similarity*, in a musically-relevant fashion? Does doing so produce a significant increase in the computational complexity of the construction process?
- Can the computational complexity of the grammar construction process be reduced by the introduction of some form of heuristic, such that grammars may be built quickly, or very complex and previously intractable grammars may be constructed in a practical manner?

Answers to these questions are important to assessing the applicability and usefulness of string-based compression, and in discovering whether such techniques may be improved in capability and practicality, in relation to musicological tasks.

A simple example Grammar

A simple melody – that of the traditional English nursery rhyme "Three Blind Mice" (Howard, 1952) – may be seen in Figure 1.1. Note that bars 1 and 2 repeat twice, as do bars 3 and 4 (although the final crotchet in bar 4 is not played until the final repeat occurs). The notes of this melody may be represented (including repetitions) as the following sequence of note names, simplified by ignoring rests and treating same-key tied notes as a single entity:

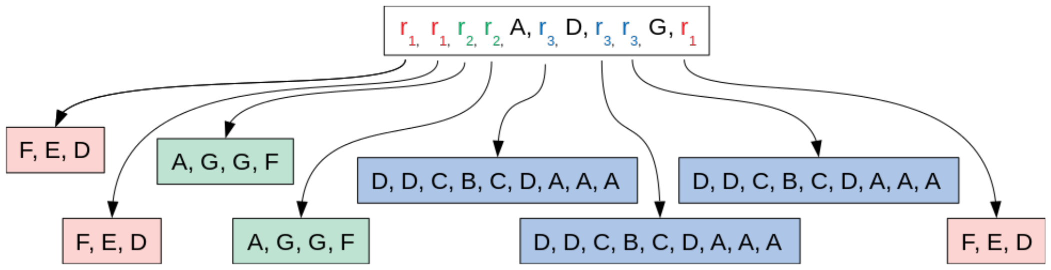

r1, r1, r2, r2, A, r3, D, r3, r3, G, r1
F, E, D
A, G, G, F
D, D, C, B, C, D, A, A, A
D, D, C, B, C, D, A, A, A
F, E, D
A, G, G, F
D, D, C, B, C, D, A, A, A
F, E, D

Figure 1.2: The simple dual-level hierarchy represented by a grammar generated from the main melody of "Three Blind Mice".

$[F, E, D, F, E, D, A, G, G, F, A, G, G, F, A, D, D, C, B, C, D, A, A, A, D, D, D, C, B, C, D, A, A, A, D, D, C, B, C, D, A, A, A, G, F, E, D]$

In turn, this list may be turned into a sequence of string characters, as follows:

FEDFEDAGGFAGGFADDCBCDAAADDDCBCDAAADDCBCDAAAGFED

If fed into a standard grammar-based compressor (Carrascosa, Coste, Gallé, & Infante-Lopez, 2011), the following output is produced, representing an encoding of the grammar generated:

$r_1r_1r_2r_2Ar_3Dr_3r_3Gr_1\$FED\$AGGF\$DDCBCDAAA\$$

In this encoding, the symbol $ represents a separation between production rules, and the rules $[S, r_1, r_2, r_3]$ are therefore distinguishable. Beginning with the rule S, and replacing any rule symbol with the contents of the rule it references, the original string shown earlier may be exactly reproduced. The input requires 47 string characters to store, whereas the grammar requires only 31; thus, the latter is a compressed version of the former. An expanded version of the encoding may be represented by the hierarchy shown in Figure 1.2:

The rules chosen by the compressor during the grammar's construction may be indicated on the original musical score, as shown in the Figure 1.3. As much of the nursery rhyme's repetition as possible has been identified, and leveraged in generating the compressed version of the note name sequence. Although musically it is arguably an imperfect segmentation, this simple example clearly shows that some significant patterns within the piece have been identified as part of the compression process.

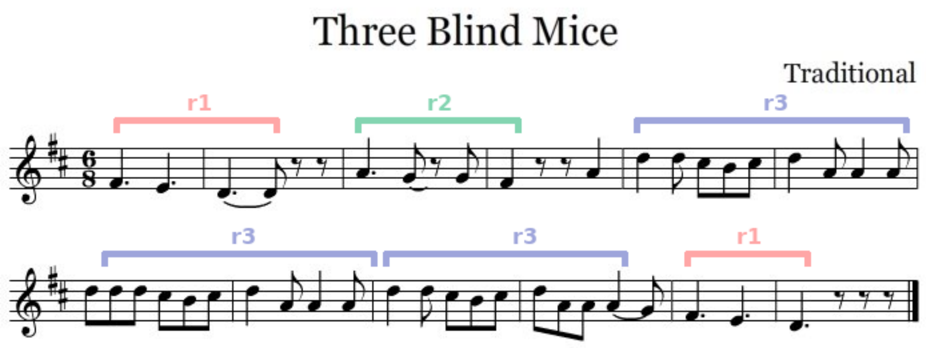


Figure 1.3: Note groups represented by the rules of a grammar generated from the main melody of "Three Blind Mice".

Generalised algorithms

Many algorithms which are designed to assist in the analysis of music either rely on additional input from a user with some knowledge of the art, or require some domain-specific prior set-up, such as those which are based on Deep Learning models and must first be trained from a large number of inputs. A simple example of the application of such "domain knowledge" may be the segmentation of a sequence of notes into likely groups of bars; to achieve a reasonable degree of success on such a task, it is necessary to understand what is commonly meant by the term "bar", and what characteristics a group of these measures is likely to present, perhaps estimated from observing correspondences and differences between a large number of example sequences. This information may then be used to predict a segmentation of the sequence which is likely to be both valid and highly probable given the known context. However, it can be desirable to employ an algorithm which either does not possess prior knowledge, or relies on the most rudimentary of rules, because such a system can offer a continually generalised response to any new input it is presented with regardless of whether it has seen an input of that character previously. As such, it is capable of immediately responding to novel data, does not cause any training overhead, and assures continued performance to a known standard.

There is strong motivation for discovering useful algorithms of this type, because of their versatility and potential for use on data which has yet to be created. A system which is capable of automatically generating a complete and valid musicological explanation may be used not only to provide an understanding of an existing musical work, but also in the generation of new material. A composer might experiment with a variety of new ideas, and examine the structural changes which result from their exploration – the availability of an automatic analysis of the score's structure could aid learning at a student level, or even provide feedback to an expert, allowing them an external perspective on their work which might be used to extend the range of known options at any given stage of composition.

Alternatively, a system which is capable of identifying information which is common to different works, which by extension may also be used to measure uniqueness, might be used to allow a user to select music which is similar to a preferred example, retrieve a specific piece from a collection by feature or segment, or test their own works to prevent accidental plagiarism and enhance originality. The user may be highly experienced, or novice; under ideal circumstances, the response of the system might be similar in quality to that of an expert musicologist, and so able to provide as much or as little expertise as needed for the application. Any algorithm which can deliver this performance is desirable, but a particularly versatile algorithm may be able to continue to deliver without any intervention even on unseen data.

Clearly, a method which does require training to be able to correctly handle novel types of data, such as a Neural Network based analyser, may be preferable if it is able to operate more quickly, more efficiently, or more accurately than a method which does not need maintenance or expansion. Thus it is important to evaluate any candidates of the latter class, to ascertain whether they are indeed useful alternatives, and what trade-off exists when selecting between the two classes. These considerations are all strong motivations for this work: the discovery of algorithms which do not include any specific representation of domain knowledge but are able to perform analytical tasks given musical data; assessment of the conditions under which such algorithms function, alongside their advantages and disadvantages; and empirical evaluation of performance on actual tasks given real-world data.

Grammars are a strong candidate structure, and their associated generation techniques offer much promise in this regard. Steedman (1984) showed it was possible to construct a generative grammar which correctly modelled known 12-bar blues sequences, along with unusual but valid variants, and more recently Sidorov et al. (2014) showed that grammar-based compression was able to generate structure highly similar to that identified by a musicologist. Whether the identified features are extracted because of their musical salience, their repetition, or both, is an interesting research question. However, no study has yet evaluated the ability of such grammars to perform musicological tasks given a large sample of musical scores, or the effect of grammar construction algorithms which can produce smaller, more optimised models on these tasks. There exists the opportunity to investigate these techniques, with the compelling possibility of discovering an algorithm without domain knowledge which is highly effective when applied to the analysis of musical scores.

More compact models

The Minimum Description Length principle supports the assertion that a more compact model is a better representation of the underlying structure and parameters of the object it represents. It follows, therefore, that techniques which may be used to further reduce the size of the encoding are likely to lead to generation of a more representative, and more useful model. Techniques which enable a "flexible" approach to the representation of substrings within the model, such as the pattern-based approach of Siyari and Gallé (2017), have demonstrated the existence of smaller models and the possibility of their discovery when greater substring similarity is enabled.

Using such techniques as a foundation, this study presents an alternative approach designed to increase similarity within the input data, as a framework within which the type and extent of equivalence may be customised in any manner which aids the application. Instead of seeking segments a, b which contain identical symbolic sequences, it is possible to instead identify those which may be transformed by some function f given a parameter T which defines how the transformation is performed, such that $a = f(T, b)$. From this premise, it is possible to define similarity in a highly custom manner. The ability to automatically select from the strongly increased possibilities such a framework provides is also explored, and a powerful optimisation algorithm, Zig-Zag (Carrascosa et al., 2011), is applied to the parallel traversal of the existing dimension of substrings, and an additional dimension of ways in which substrings may be considered approximately equivalent. This thesis demonstrates that including custom transformations between two similar strings within a grammar is possible, and limited only by the ability to encode the transform in the model, and that of the restoring program in applying it. Experimentation is used to evaluate the effectiveness of the approach in generating smaller models from musical data; performing automatic segmentation of musical scores; and to classify pieces by melodic characteristic, to discover whether an increase in performance is obtained.

Reducing Construction Complexity

Since addition of a second dimension of transforms to the existing search space of substrings which must be explored, in order to produce the most compact representation, will naturally vastly increase the complexity of discovering suitable minima, the already expensive process of grammar construction can become largely impractical given large inputs, or those with high internal similarity. Given this observation, the study introduces a heuristic which can determine which production rules are likely to be most effective during grammar construction without the parsing operation necessary to generate concrete observations, by approximating the effect a new rule is likely to have on a grammar's encoding using an easily computable operation. The technique uses an array of bits which represents the input's sequence of symbols, and generates a linear approximation of the gain a given rule may provide, allowing all candidate gains to be quickly evaluated. Once a list of gains has been sorted, it is a trivial matter to evaluate each in turn, and select the first which provides any benefit for addition as a production rule.

Since the effect and applications for grammars which allow the modification of production rules during expansion are known from investigating the production of more compact models, the final section of this study explores the effectiveness of the bitmask-based technique in generating grammars which are at least approximately equivalent in encoding length, with greatly reduced construction times. A comparison between techniques in generating both regular grammars, and those which allow rule modification, is made using standard corpora and a large, custom collection of digital scores. The results provide an interesting insight into the effect of reducing the proportion of the search space which is explored, and suggest future work which may further improve the techniques described.

1.1 Roadmap

In this thesis, a review of existing work and relevant literature is first presented in Chapter 2, covering the development of grammars and their application to the analysis of language, DNA and musical data, along with alternative approaches to computation-based music analysis.

Representations of musical scores exist in many varieties, both as traditional, physical media and as digital structures, the latter being necessary where any form of digital computation is to be performed. The choice of representation can have a significant impact on the breadth of information present in an algorithm's input, and on the quality of the results obtained, and a selection must be made which is appropriate to the specific task. Chapter 3 discusses some of these representations, and provides a definition of the representations used here for development and experimentation.

Chapter 4 provides a background for the compression techniques used within this study. A definition of Kolmogorov Complexity (Vitányi & Li, 2000) and straight-line grammars are given, followed by methods and attributes of grammar compression which are relevant to the work later described. This chapter introduces ZZ (Carrascosa et al., 2011) as an important approach to optimisation of grammar encoding length.

Experiments are performed using a large corpus of musical data in Chapter 5, including evaluation of the ability of grammars to detect and correct errors in musical scores, to classify music by "tune family" (van Kranenburg, Janssen, & Volk, 2016), and to segment scores in a manner similar to that of an expert musicologist. A novel method of error detection is described, and alternative methods of compression are used to provide a benchmark for comparison.

In Chapter 6, an extension to traditional straight-line grammars is presented which allows the encoding of transformations which can modify production rules during expansion, thereby allowing them to flexibly match in a manner which is extensible and offers the opportunity to add domain knowledge, such as the common transposition of patterns within a musical score. Challenges with the method are discussed, and an approach based upon ZZ optimisation is defined which allows a locally compact grammar of this type to be discovered. The method is explored empirically in Chapter 7, where its ability to compress, discover significant sections within musical data, and classify by "tune family" are explored.

This experimentation shows that the method results in a dramatic expansion of the search space which must be traversed during discovery of a compact grammar, and this issue is discussed in Chapter 8. Here, a heuristic is presented which produces a strong reduction in practical search complexity, and allows the fast generation of grammars which may results from the use of ZZ. The technique works by approximation of the gain each constituent can provide to grammar, creating the opportunity to perform only a limited umber of expensive computations during their selection. Chapter 9 explores the effectiveness of the method empirically, evaluating its ability to compress musical inputs both with and without encoding transformations.

Finally, Chapter 10 gives a summary of the overall findings of the work, along with separate conclusions for the experimental sections exploring the application of straight-line grammars to musical data, the addition of rule modifications, and the effectiveness of the presented heuristic in reducing occurrence-optimised (Carrascosa, Coste, Gallé, & Infante-Lopez, 2010) grammar construction complexity.

1.2 Contributions

The key contributions of this study are as follows:

1. An extensive investigation of the performance of grammar-based compressors on a range of orthogonal analytical tasks, given a large corpus of musical scores as input (Chapter 5).
2. A novel method of generating more compact grammars, by allowing production rules to become modified during expansion by versatile transforms also stored within the encoding (Chapter 6).
3. A heuristic allowing the faster construction of grammars by simple approximation of their benefit to the encoding, enabling constituents to be chosen without prior, expensive evaluation (Chapter 8).

Item (1) above covers a number of applications, which may broadly be split into the tasks of classification, segmentation, and correction of data errors. The specific applications explored are:

- Identification of a single data error within a musical score (Section 5.4.3).
- Identification of multiple data errors within a musical score (Section 5.4.3).
- A novel method allowing the automatic selection of candidate Transcription Error positions (Section 5.4.3).
- Classification of the *Meertens Tune Collections* by "tune family" (Section 5.5).
- MIREX 2016 *Discovery of Repeated Themes & Sections* task (Section 5.6.3).
- Structural Analysis of Bach's *Well-Tempered Clavier* (Section 5.6.3).
- Simple grammar-assisted editing of a musical score (Section 5.6.3).

Exploring the performance of grammar-based compressors on these tasks provides an indication of their usefulness with respect to music analysis, and a measure of the response which may be expected from grammars which model musical sequences. Introducing custom transforms into a grammar encoding enables increased equivalence between segments of the input sequence, and allows some domain knowledge to be considered during the construction process. Finally, enabling grammars to be constructed with reduced computational complexity not only allows existing tasks to be performed more quickly, but also alleviates the strong complexity which the addition of rule modification can introduce to a grammar.

It is much more rewarding to do more with less.

Donald Knuth

2 Literature Review

THE analysis of structure within a musical piece is an important aspect of music analysis, and traditionally achieved through the context of music theory. Schoenberg (1967) asserted that organisation in the form of logic and coherence is what separates random noise and musical form, comparing this structure to the application of grammar to a language. Schenker (1969) was also convinced of the existence of organisation, and developed a pitch-based hierarchical model of musical form which he used to analyse a great many works. Both authors identified hierarchical elements based on pitch and rhythm, but did not offer a scientific formalism of their observations. Nonetheless, their work highlights the ability to discern hierarchy as an important aspect of music analysis, and suggests any successful solution to this problem should take such structure into account. Many other analytical approaches exist, including those which focus on performance (Lerch et al., 2019), compositional form, and even physical gesture (Gritten & King, 2006), where a listener's physical responses may be used to map expressive dynamics to the musical surface. Although an entirely holistic approach is not unthinkable, the various techniques should be individually understood. Structural analysis, and the role of grammars in performing such analysis, has been selected from the studied techniques as an appropriate basis for this study.

Repetitive structures present within a musical piece may be leveraged to compress it. The Minimum Description Length principle (Rissanen, 1978) suggests that the best description of a given data series, such as that formed by the sequences of notes present in a musical score, is represented by the smallest model capable of reconstructing that series with the minimum error. Since repeating patterns within note sequences may be replaced by references to a single copy of each pattern, such patterns may be used to compress a sequence; this provides one method of generating a compact model, and thus, for our purposes, measuring description length.

Following this observation, it is a reasonable hypothesis that compression may be used to solve a number of musicological tasks, despite an absence of any knowledge of the musical domain: to detect and correct errors, such as might occur during transcription; to classify pieces by melodic characteristic; to segment pieces in a manner similar to an expert human analyst; and to offer a structural approach to the manual editing of music. This thesis extends the work of Sidorov, Marshall & Jones (Sidorov et al., 2014), experimentally validating the hypothesis that compression is applicable to these tasks, before introducing a custom, flexible-matching method which may be used to produce more compact grammars, and a technique capable of reducing the practical complexity of grammar construction based upon approximation of the potential encoding gain an individual production rule may offer. This dissertation aims to evaluate the relevance of these novel approaches to musicological tasks, and lays the groundwork for future improvements and development.

In this chapter, grammars are presented in a historic context, alongside their usefulness in fields such as linguistics and biology, to show their origin and applicability to the purpose of this dissertation. Computational techniques for grammar construction are discussed, along with work detailing their application to the task of music analysis, to show their suitability. Finally, alternative approaches to analysis are explored, and some of the challenges facing the methods this thesis will focus on are outlined.

2.1 Grammars

The discrimination and modelling of structure, particularly in relation to language, has an interesting and extensive history. As described by Bhate and Kak (1991), an algebraic expression of Sanskrit was presented in the 5th century BC by Pāṇini, perhaps the first example of a linguistic grammar. In 1956, Chomsky (1956) formally defined grammars as part of his hierarchical theory of languages, investigating their properties further in 1959 (Chomsky, 1959). Such grammars were designed to capture language and "phrase" structure, and allow derivation of a large but specific set of sentences, producing more realistic output than existing techniques, such as finite-state Markov models. Chomsky also showed transformations, which he defined as connections between terms which could be used to infer relationships between them, could be applied to produce more complex variations. These works suggest that grammars are highly suitable for modelling human-generated structures across a wide variety of communication styles, which in turn may make them applicable to the modelling of musical sequences. The principal component of a grammar is its production rules, each representing a segment of the symbol sequence being modelled. A single symbol, referring to a rule, may then be subsituted within that sequence during expansion to allow reproduction of the original sequence. Symbols within a grammar are composed of the output alphabet, known as terminal symbols, and rule references, known as non-terminal symbols since they represent additional content, or "branches", within the grammar's hierarchy.

Grammars have been strongly studied and applied to a wide variety of tasks, in particular within the fields of linguistics and biology. In 1964, Solomonoff (1964) explored their relevance to phrase structure induction, and a year later Bellert (1965) compared phrase structure grammars to relational grammars, showing only the latter was able to correctly generate specific Polish kernel sentences. Certainly, these studies show that grammars are able to correctly capture complex structural and relational information, and may be manipulated for useful purpose.

Sundberg and Lindblom (1976) noted great similarity between the derivation of generative grammars from both linguistic and musical sources during their 1976 study, where models which could generate a set of nursery rhymes and folk songs were explored, underlining the usefulness of grammars in the musical context. The importance of identification of hierarchical structure was highlighted, and a general method of studying both language and musical data was outlined, in which the production of a model was shown to be guided by both observation and feedback based on evaluation of predictions made.

Langley (1995) presented an alternative approach to linguistic grammars, using sets of sentences to produce an adaptive model capable of identifying and generating valid sentences with high probability. VanLehn and Ball (1987) applied version space learning to the induction of context-free grammars, to enable human-like modelling of arithmetic procedures. They concluded that although it is apparent the method could not directly emulate human learning, it performed well in their application, and allowed determination of both accepted and rejected generalisations at any point of operation. Their work did not attempt to adapt the model to different inputs, such as music or DNA strings, but instead demonstrated that the use of grammars as functional models was both practical and extensible.

Construction of a grammar with the intention of creating a minimal encoding of a sequence is a valid method for compressing it (Nevill-Manning & Witten, 1997). However, it has been shown that common grammar-based compression techniques do not produce models of the smallest possible encoding length, and are thus theoretically sub-optimal (Lehman & Shelat, 2002). Since the problem of producing a Smallest Grammar is NP-Hard (Charikar et al., 2005), an approximation may instead be sought with lesser time complexity. Unfortunately, maximising the degree to which a generated grammar approximates the optimal solution may be important in the context of the Minimum Description Length Principle (Rissanen, 1978), as structure present in a compressed model may represent structure which is meaningful in the context of the composition of its input (Rissanen, 1978), and the obtained structure may vary strongly for models with varying degrees of compression. For instance, the three sections of a Sonata may be reflected by the top level of a grammar's hierarchy as being formed from three distinct rules, each aligned with its respective section of the original musical input. A less-compressed model may not contain three such well-formed rules, the compressor having instead selected a sub-optimal combination. Lehrman et al. (2002) defined compressive bounds for the grammar-based compressors Sequential (Kieffer & Yang, 2000), Bisection (Kieffer & Yang, 2000), Greedy (Apostolico & Lonardi, 1998) and LZ78 (Ziv & Lempel, 1978), and provided theoretical arguments supporting their suggestion that far greater optimisation of grammar-based compressors is both possible and difficult. The study showed no non-NP-Hard method with polynomial time complexity may exist with an approximation ratio smaller than $\frac{8569}{8568}$ given a specific set of constraints, but suggested careful analysis of the function of existing compressors may yield more ideal algorithms. As such, the search for algorithms which generate structure as a by-product of their operation, are not excessively complex, and are heavily optimised with respect to encoding length, is likely to be of continuing interest and of practical use. Potentially, such algorithms may be important to the field of automatic data analysis.

Carrascosa et al. (2010; 2011; 2012) showed that a variety of existing grammar-based compressors performed identical steps during the construction process. These compressors differed only in score function, selecting one of three specific functions: Maximal Length (ML), where the repeating term with the greatest length was chosen; Most Frequent (MF), where the repeating term with the highest number of occurrences in the input was chosen; and Most Compressive (MC), where both term length l and frequency f were combined as lf to allow selection of the term offering the greatest reduction in encoding length when all its instances were replaced within the input. The study presented an algorithm unifying these approaches, which they termed *Iterative Repeat Replacement* (IRR) schemes, and a new algorithm able to optimise constituent occurrence, using their *Zig-Zag* optimiser (ZZ) which explored the lattice of all possible constituent combinations in a locally optimal fashion. Occurrence optimisation addressed an issue present in all IRR algorithms, and consistently smaller models were produced (Carrascosa et al., 2011) than IRR-Mx with any score function, Sequitur, or LZ78 on pieces from the Canterbury Corpus (R. Arnold & Bell, 1997), a set of text files considered representative of a wider file collection, and intended particularly for the evaluation of compression algorithms. It was also applied to large sequences (Carrascosa et al., 2012) and DNA strings, where it exhibited the same strong performance. The authors suggest the addition of inexact constituents, or removal of the lossless constraint, may be worthwhile, and in the latter case may allow better recovery of structure from DNA sequences. Their comments provide an instance of support for the research of more flexible and powerful representations of repetitions within a sequence, a concept fundamental to this thesis – specifically, a range of possible transformations might be associated with each pattern, so that a higher number of repetitions may be identified with a given string.

Benz and Kötzing's GA-MMAS algorithm (2013) used heuristics in its traversal of the same search space as ZZ to select constituents with a high probability of belonging to an optimal set of constituents, enabling it to construct grammars of equal and lesser size in fewer parses. Greedy selection of the constituent giving an immediately smaller model was not made at each lattice traversal step. Instead, a Min-Max Ant System (Stützle & Hoos, 2000) was first used to guide the traversal swiftly to selections close to the node representing the smallest grammar, where a genetic algorithm with a population of size 30 was used to choose a selection minimising the size of the resulting grammar. On the Canterbury Corpus, GA-MMAS produced smaller grammars than those found by ZZ (Carrascosa et al., 2011), and when applied to Grumbach and Tahi's DNA corpus (Grumbach & Tahi, 1994), was able to improve on ZZ models for all but three sequences. No implementation of GA-MMAS was made available by the authors. Nonetheless, the work demonstrates the utility of heuristics in the constituent selection process, and the existence of smaller encodings where an alternative path through the search space is traversed. Since the work represents the most powerful variation on ZZ optimisation to date, further focus on constituent selection and discovery of more optimal encodings is warranted.

Siyari and Gallé (2017) introduced the concept of flexible matching, where terms to be replaced within the input string were not limited to those which exactly repeat, but instead constucted from a pattern *uwv* where *w* was allowed to vary so that only the prefix *u* and suffix *v* of each pattern were equal. Their approach allowed the use of flexible patterns without an explosion in the size of the search space. The technique was also applied as a post-processing step to the output of existing grammar-based compressors, and their method was shown to produce smaller encodings for most files within both the Canterbury Corpus and Grumbach and Tahi's DNA corpus. Experiments showed that the technique was better able to identify syntactical structure in linguistic data than IRR or ZZ. In many cases, their flexible matching outperformed GA-MMAS when applied to the Canterbury Corpus, although only one DNA sequence resulted in a smaller model. The authors identified the necessity to avoid inclusion of a separator symbol between each replacement instance within a branching rule, since this factor alone increased the cost of the rule to a point where its gain was negated. This finding is significant – it highlights the importance of including processes which apparently allow a reduction in encoding size when calculating that size. They noted that a single two-element rule containing separators, although allowing each of its elements to be of varying length, could not offer better gain than selecting two individual, exactly matching rules for inclusion within a given grammar. This fact demonstrates care must be taken when selecting an encoding for any flexible matching scheme, in order for it to present any advantage to a grammar-based compressor. Most importantly, the work proves that alternative models do exist for a data sequence which are more compact – and, potentially, more accurate given the Minimum Description Length principle – yet which rely on production rules which do not always generate a uniform output. These factors suggest the existence of different types of rules which may be important to a more optimal solution. Development of such rule types is a foundational concept for the work presented in this thesis.

Overall, these existing studies show that compression can be appropriate to the modelling of data, and that grammars constructed from music – and other input types – can exhibit useful structural properties. Grammars generated by compression therefore have the potential to be an effective tool for use in analytical tasks, and there are still many aspects to their form and construction which represent interesting and promising research opportunities.

2.2 Grammars in Application to Music Data

In his review of Bernstein's lecture series, "The Unanswered Question: Six talks at Harvard" (Bernstein, 1976), Keiler (1978) warned that direct application of linguistic structures to music analysis may not be appropriate, although he believed its principles and the process of clearly formulating analytical problems in search of formalism remained important. However, Roads and Wieneke (1979) investigated the suitability of grammars to the study of music composition and structure, demonstrating that they were indeed useful: in reference to a model representative of poetry, the authors state that "a new poetry is only possible with a new grammar", and suggest great flexibility may be achieved by separation of a model of musical composition into symbolic and sonic components, whereafter the grammar may generate a set of musical scores. Discovery of the extent of the relationship of language to music, and how practical use may be made of this knowledge, is clearly an interesting and potentially important area. Recognition of the individual, often interacting layers present within a composition is also important, and a reflection of the hierarchical approach taken by musicologists to score analysis.

Many studies have suggested formalisations of generative musical grammar. Steedman (1984) presented a music-orientated example of such formalism in 1984. Referencing the ability of musicians and non-musicians alike to evaluate whether melodies are well coupled to the piece to which they belong, he suggested that a chord sequence forms the foundation which enables such distinction to be made, and from which a model representative of a piece may be created. He outlined a generative grammar modelling various sequences of the traditional 12-bar blues, and showed it were possible to derive new variations on this structure, pointing out that additional rules might allow for a more refined definition. Although admittedly based on a relatively simple and tightly-specified form, the work showed a generative, hierarchical model could indeed represent real-world musical structures.

In 1968, Winograd (1968) explored the application of Chomsky's linguistic techniques in order to model the specific rules and structure which may be observed in the majority of musical works. In particular, emphasis was placed on harmonic structure as described by Forte (1974), and a heuristic-driven algorithm was developed, allowing production of a model generally applicable to a given musical piece or type. Winograd observes that "a complete parsing system for music would be extremely complex", and highlights the significance of interplay between different compositional factors, such as pitch, inversions and time signatures, along with the existence of multi-dimensional, independent structures. His work highlights the need to model the more complex interactions occurring within a musical composition if an attempt to construct its complete model is made, and the potential for these interactions to occur in an overlapping manner. It is possible that domain-specific knowledge must be integrated into such a system, and a universal approach may not entirely capture the closely-coupled factors he observed.

Ulrich (1977) applied generative grammars to the task of harmonic matching, computing the most probable chord and key for each step of a chord sequence, and showing how this may be used to fit abstract melodies to a given piece in a musically acceptable fashion. This work was intended as a limited demonstration of the computational analysis of music, and Ulrich's algorithm selects the model containing the fewest key changes as the most likely representation of chord sequence, in contrast to Winograd's use of heuristics. However, it offers evidence supporting the assertion that a grammar may realistically represent the underlying structure of a piece, and parameter minimisation as a standard optimisation technique may prove useful in discriminating between possible versions of a model.

In 1997, Nevill-Manning and Witten presented Sequitur (1997), a grammar-based compression algorithm with time complexity $O(\sqrt{n})$. It was applied to an English novel, and the authors demonstrated that potential compression was directly related to information entropy. Their method was shown to compress large DNA sequences more effectively than competing algorithms, and segment text hierarchically in a meaningful manner, despite having no prior knowledge of either subject. It was also shown to be able to correctly select repeating motifs from two Bach chorales, although its performance in this regard was not extensively or musicologically assessed. Despite this, Sequitur is often referenced in connection with the ability of a compressor to discern phrase structure in music, and proves at minimum that musicologically-significant phrases may be recovered from music data by methods which are not constructed specifically for that purpose. It also shows that a computationally complex algorithm is not necessarily required to achieve this, and it is possible for a more practical, less optimal process to have relevance to the task. It is probable that investigation of response on a large, representative population of musical works may yield useful insights in this respect.

Abdallah, Gold and Marsden (2016) took a probabilistic approach to structural analysis by grammar, and provided an in-depth review of related methods. Grammars were also used to model a corpus of musical scores, from which they claimed "sentences" characteristic of the corpus were generated. They suggested the form of a grammar intrinsically models musical segments, the classes to which they belong, and their relationships. Their study provided an overview of techniques for performing inference, music analysis and grammar construction, alongside a demonstration that their models were able to effectively segment symbolic scores within a dataset of Bach chorales into groups of repeating symbols, represented by the right-hand sides of the grammars' rules. They noted, however, that Markov models remained superior when applied to a corpus of folksongs gathered by the University of Essen (Schaffrath & Huron, 1995), although these lacked the structural benefits of grammars. The paper did not evaluate the musicological correctness of the obtained models.

In 2014, Sidorov, Jones and Marshall (2014) presented a concise study showing that straight-line grammars formed from sequences of voice intervals could be applied to tasks such as editing and error detection, and suggested they might also be useful for summarisation, simplification, similarity estimation, and plagiarism detection. Single-symbol data errors were introduced to successive intervals within Bach compositions, and the resulting model size variations recorded, producing a response which generally increased; the authors attribute this to degraded structure, causing the data to be less compressible. Rules within a grammar hierarchy were shown to map well to a human-defined musicological segmentation, although the authors did not investigate correlation with accepted expert analyses. The grammar construction algorithm used, whilst effective, was not optimal in constituent selection, potentially resulting in an undesirable increase in the size of some models. Nonetheless, the authors presented a compelling case for the relevance of minimal grammars to the analysis of music.

In 2020, Mondol (Mondol, 2020) demonstrated the suitability of the Context Free Grammar as an estimation of Kolmogorov Complexity, using models constructed from individual and concatenated musical pieces to generate pairwise normalised distances from which classification by genre, composer and style was accurately performed. This work was extended (Mondol & Brown, 2021) by using specific candidate production rules from a model in the compression of other inputs, showing the approach was most successful where both inputs were created by the same composer, thus providing a measure of similarity between pieces. The study provided strong support for the hypothesis that the rules of a grammar are able to capture information which is structurally significant to a composition.

The works referenced in this section often demonstrate that, as noted by musicologists and composers such as Schoenberg and Schenker, it is probable that musical scores contain patterns which interact on the musical surface in a complex manner, but equally that even relatively simple approaches may yield surprisingly useful results with a musical context. Further study in this direction may include searching for methods which are able to correctly identify and model these patterns and interactions, and should not necessarily exclude algorithms which are not optimal in terms of compression. However, more information is needed to help quantify the performance of grammars and compressors with respect to music analysis, and whether compression strength must be optimal in order to have real-world applicability across the population of musical inputs.

2.3 Algorithms for Music Analysis

The previous section focusses on the application of grammar-based techniques to the tasks of identifying potentially useful or significant structures within music data, and of deriving a concise representation of any given musical score. However, much of the research conducted into approaches for the automatic analysis of music is aimed at a more specific goal, and often targeted at a specific theory or methodology. Some notable examples are discussed within this section.

Lerdahl and Jackendoff presented *A Generative Theory of Tonal Music* (1983), where they discussed two complementary aspects of analysis, structural correctness and rules of preference, before presenting a defined, hierarchical method. There have been several strong attempts to implement their techniques computationally, including an interactive analyser (Hamanaka & Tojo, 2009) which outperformed a purely automated approach, a variety of analysers (Hamanaka, Hirata, & Tojo, 2016) designed to overcome theoretical difficulties, and a deep learning approach (Lai & Su, 2021) trained on a large, algorithmically-generated dataset which outperformed existing boundary detectors based on grouping preference rules. Alongside showing that the task of music analysis may be conducted computationally but based upon a human-defined process, these contributions also draw attention to the wide range of techniques which can be successfully applied, including neural networks, fixed rules, and human-guided processing.

Many other algorithms have been developed specifically for application to music data. Building on earlier work (De Haas, Rohrmeier, Veltkamp, & Wiering, 2009) and implementing a modified representation of harmonic structure based on generative grammars (Rohrmeier, 2007), De Haas et al. presented the Haskell-based HarmTrace (2011), which was capable of producing an analysis of harmonic progression given a sequence of chord labels. It was unable to delineate phrase boundaries, although the authors suggest this may have been possible as additional processing, and it was shown to perform well on harmonization, chord recognition and harmonic similarity tasks. The earlier study (De Haas et al., 2009) showed inclusion of musical knowledge was able to improve performance for a similarity-based retrieval task, suggesting domain knowledge may be important to algorithmic analysis. As stated in a later journal article (De Haas, Magalhães, Wiering, & C. Veltkamp, 2013), the use of grammars in this context allows generalisation over a large state space, and automation of conditional rule selection. Inclusion of domain knowledge in straight-line grammars was not part of these studies, and may be a promising area of investigation.

Rohrmeier (2011), extending a study proposing a phrase-structure grammar (Rohrmeier, 2007), provided further support for the existence of discernible recursive and hierarchical structure within music, and offered a review of work demonstrating that contextual importance and purpose may be attributed to logical musical progressions. The study presented a practical formalisation of a generative grammatical model capable of explaining the harmonic structure of an extensive number of varied works, where units such as chords and notes were themselves derived from tonal elements. Although drawing strong parallels between music and language, the temporal nature of harmonic relationships is highlighted as being worthy of further investigation, and caution against assuming a direct correlation is given. Rohrmeier states in reference to Markovian models:

> "Transition matrices or n-gram models may well merely reflect statistical properties of underlying more complex deep structure processes (Rohrmeier & Cross, 2008) and therefore cannot be easily argued to constitute 'the' structure-building process (pace (Tymoczko, 2003)). By themselves, they do not embody sufficient complexity to express the formal structure of harmonic tonality, modulation processes or overarching formal processes."

An automated approach capable of modelling such generative processes is clearly desirable, and the author does not discount the potential of statistical modelling as a way to harness musical structure, although the importance of hierarchy and abstraction in understanding harmony is demonstrated and discussed. The study does not itself suggest a scheme making use of straight-line grammars may be successful in discovering the identified processes, but does offer support for the validity of further research into structural approaches for harmonic modelling and analysis.

Some notable studies which explore statistical or rule-based approaches to the extraction of information from symbolic music have been conducted. Temperley (2001) developed a "preference rule" system which aimed to identify high-level structure within 18th and 19th century tonal music, using rules based upon six specific attributes: metre, phrase structure, counterpoint, pitch spelling, harmony, and key. These were based on empirical observations from the field of psychology, alongside analytical theory. The scheme was applied to music which was performed with an expressive interpretation, as well as unmodified scores from which more conventional compositional elements may be discovered, although only an elementary musical structure overall was sought given the constraints of the chosen parameters. A simple "piano-roll" representation was employed, with timbre and amplitude information discarded from the input data, although the system still performed effectively, suggesting these attributes were not critical to the tasks being investigated. Dynamic programming was discussed as a method for selecting between competing interpretations, highlighting that optimisation may be usefully employed where ambiguity exists. The work was aimed towards higher-level abstractions of musical perception, and outlined a belief that minor changes to an expertly-constructed score – for example, the alteration of single note – would not necessarily be of significant detriment to the piece. This supposition was not explicitly tested, although it is reasonable to assume the presented system may prove invariant to such changes, perhaps unlike a more granular method. Overall, the publication provided an accessible and comprehensive introduction to the cognition of musical structure, and rule-based computational approaches to the modelling of human analysis.

Temperley (2007) also applied Bayesian reasoning (Berry, 1996) to the identification of musical structure, and presented three probabilistic models. The first inferred metric structure given a monophonic sequence of note onset times, by the fitting of a grid which maximised both the probability of a rhythmic pattern existing within it, and the likelihood of the grid's existence given the specific pattern. A dynamic programming scheme was presented to perform the optimisation. Initial parameters were either empirically estimated from a collection of scores, or manually chosen, showing that a limited degree of context was required for correct operation. The second model inferred a key given a monophonic sequence of pitches, using assumptions for the likely distribution of the pitch range of a melody, the pitch interval between adjacent notes, and the major or minor scale degrees in use, and selecting the key for which all probabilities were maximised. Once more, initial parameters were required to be set, and these were estimated from the analysis of a score collection. Performance was shown to be as strong as existing key-finding algorithms. A third model inferred a key, and key modulations, from a polyphonic score represented as a sequence of unordered lists of scale degrees. The model sought to maximise both the probability of a key for the piece, and of the likelihood of a sequence of keys existing, based upon the anticipated key. Musical context was provided by the extraction of a set of scale degree frequencies from a corpus of 19th century music, and a dynamic programming scheme was once again presented as an optimiser. The model did not perform as effectively as Temperley's earlier preference rule system. The author suggests that Schenkerian Analysis might be evaluated "as a theory" using similar probabilistic models, and also postulates that *Occam's Razor* (Sober, 2015) might be appropriate in further evaluating models which represent competing interpretations for a given piece of music.

Pearce (2018) applied statistical learning to simulation of the human response to music, in particular with regard to the expectation a listener may have of events which will occur at any point in a given musical score, which may be based on a wide variety of psychological factors. The probabilistic model *IDyOM* was presented; individual instances were trained in an unsupervised manner on varying collections of digital scores, each representing a cultural perspective to which a listener may be exposed. The concept of enculturation was central to the study, which asserted that an individual's perception depends on the probabilistic prediction of upcoming musical events, based upon probabilities which are learned from their perception of regularity. *IDyOM* is a dynamic, variable-order Markov system, containing individual short- and long-term models, the former being progressively trained from the repeating structures within the piece being processed, the latter pre-trained on a large corpus of music and thus possessing some knowledge of the musical context. Although the inputs to *IDyOM* are composed only of a single representation – for example, note sequence index, pitch interval, or note duration – models may be combined in a similar manner to that observed of human listeners when identifying musical structure. Experiments showed there was good correlation between the output of *IDyOM* and the expectation, uncertainty and emotional response of listeners to a range of musical pieces. The study provided compelling evidence that probabilistic models may prove effective when applied to musicological tasks, and that statistical regularity within musical data may be leveraged to simulate a human-like recognition of structural boundaries.

IDyOM was also included in a comparison of statistical and rule-based models (Pearce, Müllensiefen, & Wiggins, 2008), where a range of systems capable of segmenting a piece of music into melodic groups by the delineation of boundaries were individually assessed on a common scale, and their outputs later combined with the intention of harnessing the optimum performance of each. The task was defined as identification of human-annotated phrase boundaries from a subset of the *Essen Folk Song Collection* (Schaffrath & Huron, 1995), where non-hierarchical phrases cover all events within each melody of the set. The studied algorithms included *Grouper* (Sleator, Daniel and Temperley, David, 1999), the *Local Boundary Detection Model* (Cambouropoulos, 2001), and a Grouping Preference Rule parser (Frankland & Cohen, 2004). Of all the systems tested, *IDyOM* represented the only method based on statistical, unsupervised learning. The combined model achieved a higher F1 score overall, and, notably, *IDyOM* performed relatively well in comparison to the rule-based methods, showing that the inclusion of explicit musical knowledge is not mandatory when segmenting a melodic sequence in the manner of a human expert.

Marsden (2012) examined melodic similarity, concluding that human opinion was so widely varying that it may be considered non-deterministic, although the sample used was too small to allow definite identification of salient factors. A previous study A. Marsden (2010), where the space of possible Schenkerian reductions was shown to be computationally large, is cited as an example of the significance of coincident, valid interpretations. The work suggested the presence of a third melody may affect the perceived similarity of an existing pair, and that a model capable of multiple interpretations given different data combinations may be a suitable definition of the reductive space. Here, the "structure-building process" as described by Rohrmeier (2008) directly represents the intention of the composer; the paper shows how Mozart made explicit use of the human tendency to seek similarity within his String Quartet in A Major (K. 464) and C Major (K. 465), and as such discovery of this process requires context, and cannot be understood from a single-viewpoint reduction. The study underlines a challenge for the evaluation of systems which generate a single output when attempting to perform structural analysis: a large number of possible outputs may be entirely valid, but each may require validation by expert as confirmation, making automatic assessment of performance non-trivial. However, it also highlights the relevance of similarity to musical compositions, which suggests a useful analyser should take repetitions and variations into account as part of its function.

Combinatory Categorical Grammars (M. Steedman & Baldridge, 2011), which are lexicalised linguistic grammars, have been used to model the harmonic structure of music (M. T. Granroth-Wilding, 2013). In this approach, statistical models which were able to leverage historic parsing results were used to separate syntactic structure from semantic, hierarchical chord sequences over a manually annotated corpus of jazz pieces. Chord sequences were defined via grammatical modelling. This work emphasised the importance of such separation, highlighting where previous studies had failed to consider these elements individually. The authors demonstrated the superiority of their method to Markovian analysis, and also showed how note data from MIDI performances may be used as input to generate harmonic structures. They liken these analyses to the task of sentence interpretation in natural language processing, and suggest the high degree of ambiguity to be found within long sentences is parallelled by musical constructs. Their study provides convincing evidence that statistical modelling is capable of recovering meaningful information from raw musical data when applied with consideration, and specialist grammars may make good hierarchical descriptors given a suitable input.

McLeod and Steedman (2017) presented a grammar-based method of meter detection within monophonic symbolic music, using a combination of lexicography and probabilistic context free grammars. Lexicalisation was applied to CFGs so that note durations could be considered in a relative context; this was achieved by associating the value of the longest note duration in each subtree of a grammar with its head. The probability of the correctness of various metrical interpretations for a given input was then used to generate a valid segmentation of the musical sequence at a bar level. The method exhibited stronger performance than a simplistic guessing scheme, or a non-lexicalised CFG approach, showing that significant and often complex dependencies exist between the rhythmic components of a score.

In a subsequent publication, a combination of NLP-based statistical learning and hierarchical grammar (M. Granroth-Wilding & Steedman, 2014) were again shown to be more effective than a Markovian model. Interestingly, the study proposed that difficult chord sequence analyses may be achieved by first isolating structures which are identified with high confidence, allowing a less constrained search to model the remaining segments. Additionally, it suggested that a desirable extension might be to perform prerequisite tasks, such as identification of prominent notes and segmentation into musically appropriate passages, as part of the analytical process. Given that it is possible to infer parameters inherent to the construction of a data series from a minimal model of that data (Rissanen, 1978), a compressive modelling technique capable of representing musically significant parameters, such as dominant note or pattern occurrence within a hierarchy, might present an opportunity to perform this type of comprehensive analysis automatically.

Although the above approaches vary widely in technique, aim, and outcome, there are several common elements to be found. Perhaps the most notable is the complexity of the compositional process, and the great space of valid interpretations as defined by human perception. The works suggest a truly successful approach to music analysis, which is able to account universally across genre, style and form, would not only be inclusive of a large number of concepts, contexts and compositional devices, but would also be likely to generate a wide range of different outputs, each correct from its own perspective. No unified theory of music which explains the vast array of known compositions currently exists, and potentially the attempts which target specific analytical paradigms, such as the *Generative Theory of Tonal Music*, may be most successful, or at least simplest to demonstrate as such. Nonetheless, if a less holistic approach is undertaken as foundational work building towards a unified system capable of automatic analysis, these works support the potential of hierarchical, statistical or similarity-based modelling to discover relevant properties and structure in musical scores.

More specifically, the present thesis focusses on the application of string-based compression algorithms to digitally-represented musical scores, and exploration of other promising approaches – statistical learning, rule-based models, or neural networks, for example – is considered outside of its scope. There are several reasons for this decision. A model which does not require any training would be beneficial, since it would be able to respond in a generalised manner to previously unseen inputs, and an understanding of its operation might in turn enable a more formal explanation of the process of music analysis. It may also be possible to identify areas in which musical knowledge is explicitly required by examining where such a technique fails, and an exploration of the effectiveness of any method which does not require preparation or pre-training is likely to provide knowledge which is useful in guiding future development decisions. Much is known regarding the creation and performance of probabilistic and rule-based models, and examination of the manner in which these operate individually is helpful in creating hypotheses regarding how the algorithmic analysis of music may work generally. This would, of course, also be true of any additional exploration of compression-based modelling. As discussed here, much existing work shows that the inclusion of human-defined musical knowledge is beneficial to algorithms which retrieve information from musical scores, but may also result in an inability to correctly process music whose attributes or style fall outside of that definition. Other work highlights the existence of techniques which are able to use machine learning to develop musical knowledge. As a relatively new approach which does not necessarily require such knowledge, the depth to which string-based compressors may be used in the generation of grammatical models which expose musical information is not yet known, and an exploration of this is likely to result either in an additional method of computational music analysis, or to provide an insight into where existing techniques may be improved or parallelled by alternative or adapted methods. As such, this thesis chooses not to pursue the development or improvement of machine learning approaches, and instead seeks to evaluate whether grammar-based compression, as a general and hierarchical model of arbitrary data, may be realistically applied to tasks which are useful in the computational analysis of music.

2.4 Conclusions

Music itself, and study of the art, is ancient and historic. A great many approaches exist to music analysis, which aims to understand and define a composition and the act thereof, ranging from examination of the psychological effects a performance induces in its audience, to formal, structural and mathematical approaches, many seeking to create a framework within which all music may be decomposed. Repetition has a general relevance to the generation and enjoyment of music, and is often used as a compositional tool, frequently in combination with other functions which allow variation to occur, adding texture, progression and interest to a work.

As hierarchical and structural models which can be parsed to produce an output, grammars have been shown to be useful the modelling of musical forms and individual pieces. Such models are capable of identifying patterns and structure which expert musicologists consider most significant as a result of their analyses. Straight-line grammars are suitable for this purpose, and several well-known methods exist for their construction, including Zig-Zag, an optimisation technique which is capable of selecting a combination of constituents which produce a highly compact grammar when used in combination with a graph-parsing process which outputs the most compact encoding possible given that combination.

Methods have been published which seek to produce more compact models, either through wider exploration of the search space of possible production rules, or through modelling the similarity of substrings occurring in the input by allowing sections which are related but are not exactly equivalent to match each other at a reasonable cost. However, no large-scale study of the effectiveness of these methods when applied to musicological tasks has been conducted, and compositional devices frequently seen in the musical domain have not been added to straight-line grammars with the aim of automatically producing more compact models. As such, the potential improvement which such a reduction might offer in a grammar-based musicological task is largely unknown.

Given that much work continues in the field of music information retrieval, further knowledge in this area is desirable and can help to highlight circumstances where grammar-based techniques can prove advantageous over existing methods, as well as delineate where they may fail or prove inferior to the alternative approaches. On the basis of the findings of the research discussed within this chapter, this thesis will evaluate the effectiveness of straight-line grammars on a range of analytical tasks given a large corpus of musical scores, explore ways in which substring similarity may be increased in a musically-relevant manner, and detail an approach by which the production of such grammars can be temporally optimised.

The following chapters will discuss suitable representations and musical data, and define straight-line grammars within the context of this study, before exploring how they may be manipulated and improved with respect to musical data and analytical tasks.

There can be no "true" representation just as there can be no closed definition of music.

Roger B. Dannenberg, 1993

3 Representing Musical Data, & Digital Corpora

There are a great many ways of representing music, but all may be broadly categorised by two classes: performance (Godlovitch, 2002), and notation (Apel, 1961). Performance is often a live in-person rendition by one or more musicians, recorded onto some medium from which it is later reproduced, or a combination of both. Notation instead relies on representing the music as a physical artifact, usually written, intended to be followed by musicians during later performance.

There exists a wide variety of ways in which music may be physically represented, from the hand symbols of Kodály's *Solfrège* (Dobszay, 1972) to simple notation such as recorded early *Plainsong* (Sachs, 1948), or from traditional scores (Apel, 1961) to digital archives composed of files in formats such as MIDI (MIDI Association, 1996), MusicXML (Good, 2001) or ABC (Oppenheim, Walshaw, & Atchley, 2011). Each representation has its own rules, attributes and advantages, and, regardless of its form, any study which endeavours to process music must employ some means of correctly interpreting its chosen sources, potentially translating representations which are not immediately suitable.

As discussed in Chapter 2, there appears strong potential in the use of straight-line grammars as compressed models for symbolic music, and the properties of those models may be used to achieve many musicological tasks. It is the intention of this study to examine the general response of grammar-based analytical methods with regards to a range of these tasks, and, as such, a sufficiently large population of musical works is required to enable a reasonable approximation of the universal response to be measured. Given the relative ease with which digital representations of symbolic scores may be obtained for research purposes, the potential to select between any available scale types they contain, and their symbolic, digital nature negating the need for optical recognition (as would be necessary for printed or written manuscripts), digital musical scores are selected as the source for this study. The term "digital scores" is used generally throughout this dissertation to refer to symbolic musical scores which are represented in a format suitable for storage within a digital medium, and containing sufficient symbolic information to allow the production of a full, printed manuscript, suitable for professional musical performance. Such representation also enables a subset of the information to be extracted, which may then be algorithmically processed.

A great many file types exist which are suitable for the storage of digital scores, and, within them, a wealth of forms through which the data itself may be represented. In this chapter, various methods of representing musical information in a form suitable for compression by grammar are discussed – in particular, symbolic sequences – since this is a requirement of the study as defined above. The relevance of each to the work's intentions is highlighted, and the various representations which are applied in later chapters during experimentation are defined, aiding reproducibility and an understanding of their benefits and limitations.

This chapter begins with a general description of score attributes and digital formats, and continues with a discussion of abstract representations suitable for processing and experimentation, before defining the study's chosen musical datasets and their collective properties.

3.1 Musical Scores

The terms "musical score" and "manuscript" commonly refer to a specific form of written musical representation, whose interpretation is taught worldwide. The method was originally designed to standardise written music so that is could be accurately reproduced by different religious groups (Hall, Hall, Battani, & Neitz, 2003). It is still the primary method of sharing pieces in a written form between performers, and remains popular due to its ability to capture not just the pitch-time information within a musical work, but also emphasis and context. A musical score may contain indications of dynamic (such a markings for "piano" for quiet playing, or "forte" for loud), provide a quick visual cue to the harmony of other performers in an ensemble, or may include lyrics, speech or scripting cues where the music accompanies a non-instrumental performance.

Ideally, a score will contain only information which is in some way vital to the correct interpretation and performance of the music it represents. Despite this, it is possible to examine different facets of a score, either individually or in concert, to obtain useful information. An example of this is Schenkerian reduction (Schenker & Salzer, 1969), a common task during musical analysis (Cook, 1994): some aspect of the underlying structure of a musical piece or segment is identified by reducing its complexity, commonly by removing elements so that those which remain represent the simplified structure upon which the piece or segment is based. For instance, given a sequence of several bars or notes, it is often possible to arrive at a small set of notes which represent the root pitch at any given point in the sequence. It is possible to make valid musicological observations given a subset of the information a score contains. Note that such reduction, for the purpose of analysis, is distinct from orchestral or piano reduction, where a score is reduced in complexity so that it may be performed on a single instrument whilst retaining as much of its original character as possible (so that a piece written for a full symphony orchestra may be played by only a pianist or organist, for example).

It is also not uncommon to discover that certain information within a score must, on occasion, be inferred. A good example of this is the assignation of voice (Apel, 2003), defined as a single melodic thread playing within a polyphonic ensemble, and commonly assigned to a specific musician, instrument, or group of instruments, resulting in the part's possession of a discernible character during performance. According to Purwins et al. (2008), a voice may consist of a temporal sequence of pitches which belong perceptually to a greater entity, and a voice of great salience may be considered a primary melody of a piece. Voices are sometimes specified explicitly, such as the separation of an orchestral work into individual scores, one for each instrument or section, but they may also be left for the performer or transcriber to identify using their highly-developed sense of musical perception. The task of voice inference is non-trivial (A. Marsden, 1992), and research has not yet produced a definitive method to accomplish it, although it remains an area of interest. Creative interpretation of the notion of "voice" by composers prevents a strict definition: a note which is assigned to a particular musician can be considered as belonging to their instrument's "voice", but it may also belong to a melody which may be played, entirely or in part, by different groups of instruments, often moving between them as a performance progresses.

3.1.1 Attributes of a Score

A stored representation of a musical score, regardless of its medium, may be abstractly viewed as a hierarchical composition of several elements. For the purposes of this study, the following terms are defined to represent those elements, and will be used throughout the thesis where stored score attributes are discussed. The defined terms are loosely based on those which apply to traditional Western manuscripts (Apel, 2003), digitally encoded music (A. A. Marsden, 2007; Nápoles, Vigliensoni, &

Fujinaga, 2018), or to the general perception of music (Carpenter, 1967), and are intended to form a simple, abstract framework which applies with equal relevance to an original manuscript as to a musical piece represented within a computer's memory as a set of data structures. They are not intended as a canonical definition, and are distinct from the over-arching facets (Downie, 2003) defined as significant to the domain of *Music Information Retrieval*. The following elements may be considered important to any musical representation, and represent decreasing levels of abstraction:

- *Container:* This forms the boundaries of the score, and the foundation upon which the representation is constructed. It may be a physical, visually-represented score or a digital encoding, such as a manuscript or MIDI data file, and may contain one or more pieces or "movements" (such as may be found in a multi-part composition, or a composite collection of works).

- *Surface:* The format of the score, which dictates the type and context of the musical objects within it. For written, Western musical works, this is most commonly composed of staves, upon which the "context" (e.g. staff type, key signature) and "body" (e.g. notes, chords) of the music are written. For digital scores, context and body may be stored as collections of data elements from a defined framework, such as the MIDI standard (MIDI Association, 1996). The style of their visual representation may also be specified, as is possible within Sibelius-format scores (Avid Technology Inc., 2011).

- *Structure:* Explicitly specified, implied or underlying groups, segments or delineations within the score. Form (Caplin, 2001) is an important example of such structure; for example, the classical Sonata is conventionally composed of three specific segments, beginning with a primary "pattern", which is then transformed / developed, and finally repeated and resolved, but it may be left to the performer to identify these segments in practice. A score may contain visual indications of its structure, such as repeat symbols or brackets which mark segment boundaries. These may be seen as "sub-containers" which belong on the musical surface, and are associated with its *objects*. It is at this level of abstraction that a composer may manipulate patterns and segments to produce a score whose performance is an interesting and engaging experience.

- *Objects:* Primarily, *Musical Objects*. These are the "building blocks" of the music; examples include clef symbols, notes and rests, each representing concepts such as current state (e.g. musical key) or event (e.g. the absence or playing of a specific tone). Other objects may also be present on the surface (or associated with its contents), such as lyrics or written cues.

- *Object attributes:* Each object is likely to have an attribute attached to it. For instance, where a key signature is present in a Western score, this is denoted by marking the notes which much be sharpened (+1 semitone) or flattened (-1 semitone) to produce the correct seven-note scale; the placement of these markings may be computed given the root key and scale type (e.g. G major, which contains F-sharp), and as such only the value of the root itself needs to be stored to be able to reconstruct the score when digitally represented. Conversely, when a visual score is produced, the key signature markings are placed on their relevant stave lines, and the root itself is not recorded. In this case, it is up to the performer to work out the root key if they wish, which may be done by examining the markings and using their knowledge of music theory to identify the only possible value for the root in question.

It is worth highlighting the strong presence of context, which, by tradition, is a part of human-designed musical representations. Unlike linguistics, where a particular symbol is associated with a limited set of possible interpretations or pronounciations, the symbol for a musical note may represent any of the available tones within a given pitch system, depending on its placement on the musical surface. There may also be interaction between instruments or melodic passages, such as "call and response" or harmonisation, but these interactions may be designed to occur in a fluid, changing manner, so that they appear to alternate between dominant and supporting roles within the music. Such flow can be directed by cues present on the musical surface, such as indications of timbre or volume, but the composer may intend the performance to be interpreted in a specific style. Overall structure may also be left to the performer to interpret, with the aim of either leveraging their knowledge of the musical domain to make informed judgements, or providing them with a range of possible options from which they may choose artistically to create a "unique" performance. As such, when attempting to create or model a musical score, domain knowledge may be required to arrive at an acceptably accurate interpretation. However, as with linguistic data, it is possible to exactly copy the layout and symbols of a given work without a loss of context, and leave the more difficult task of its interpretation for a future time. As such, musical scores may be converted between mediums and formats, providing all the significant components of the score are recoverable, and adequately represented in their new container.

3.1.2 Symbolic Digital Scores

Storing musical scores in a digital form can offer many advantages, although the practice can also create new difficulties (Smith, 1999). The longlevity of digital media continues to be discovered, and conversion to a digital encoding is often a costly process, both financially and in terms of resources. Where storage space must be kept to a minimum, or post-processing is intended, visual or contextual interpretation of the input is often required, such as optical music recognition, which is itself a challenging task (Bainbridge & Bell, 2001).

Nonetheless, there are sufficient positives to ensure the continued efforts of archivists and individuals in creating digital representations of new and existing music. Such advantages include preservation attempts: as the physical material that a musical piece is written on degrades with time, it is necessary to copy its contents to a fresh medium, on which it may continue to be stored within a collection or archive. If that storage medium is digital, the music may be duplicated, distributed and manipulated with ease, and there is an opportunity to apply non-destructive restorative techniques such as error correction to help safeguard the data. It is important, however, to prevent against medium obsolesence; providing it is legible, a musical score may always be read by a knowledgable human; by contrast, a digital score requires a compatible media reader and display equipment before it is viewable.

Although it is of course possible to digitise a written or printed score by taking a bitmap scan of its physical form, possibly adding compression to reduce its size, it is more common to use optical recognition techniques (Byrd & Simonsen, 2015) to identify the elements present within it, and represent these as encoded objects. This greatly increases the potential for editing and manipulation, and also reduces the storage space needed by recording only the identities and positions of the objects required for its reproduction, and not their physical images. Of course, music which has been composed since digital score editing became available may use this representation as their primary form. Scores whose elements have already been converted into their symbolic representations are the primary target of the research documented within this thesis.

Where a model of the music data is chosen as the container, as opposed to a simple graphical image, it is necessary to define the structure and attributes of the model (its "surface"), so that it may be read and manipulated, and potentially converted back to an image to be printed or displayed for a performer. There are many model formats in common use, including the following:

- *MusicXML (Good, 2001):* An open format designed to represent Western musical scores, which follows the XML (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2000) specification, enabling the use of standard parsers to extract score data.

- *Sibelius (Avid Technology Inc., 2011):* Although Sibelius itself is a piece of notation software developed and marketed by Avid, it features a file format capable of storing a full manuscript layout alongside the musical score. Some applications are capable of importing Sibelius files, but it is not an open format. Sibelius itself allows the export of files in other formats, so music stored in this manner may still be manipulated with relative ease.

- *MuseScore (Watson, 2018):* Like Sibelius, MuseScore is a piece of notation software, developed as open-source and provided free of charge, complementing the sheet-music sharing website of the same name. Although scores can be stored or imported in a variety of formats, MuseScore incorporates an open-source, native encoding capable of specifying a manuscript's layout in addition to the music it contains, making it a powerful open-source alternative to existing notation formats.

- *ABC (Oppenheim et al., 2011):* This text-based format was designed for easy interpretation by both humans and computers. It is composed of a header and set of notes, and symbolic labels which denote the meaning of particular fields within the file. It does not carry typesetting information, unlike MusicXML or Sibelius.

- *Humdrum ("**kern") (Sapp, 2005):* This is also a text-based format similar to ABC, and uses numbers to encode note durations. It also contains only the "functional" components of a musical score, without any typesetting instructions.
- *MIDI (MIDI Association, 1996):* An event-based format for storing musical information. Each event is composed of a set of attributes including onset time, note number, and velocity, and only minimal contextual information is stored, such as tempo and meter. Principally, MIDI data is designed to directly control instruments, instead of representing a score to be read by a human performer. Despite this, MIDI remains a highly popular and portable format for sharing music data.

3.2 Musical Objects

Since the same piece of music may be represented by any of the above models, it is unsurprising that many musical objects are shared between them. These objects may be broadly categorised into those which define the context of the surface (for instance, clef markers or key signature), and those which define the events which are present within the piece (notes or chords, for instance, or rests where no additional notes are to be played). A note may consist of the following attributes:

- *Pitch:* The key which the note refers to. In Western music, this may be *chromatic* and associated with one of twelve notes into which each octave is divided equally, or it may be scale-based, where all notes within the octave may still be represented, but in reference to a reduced number of notes per octave. For example, a *diatonic* scale has only seven notes per octave, but those which remain from the chromatic scale can still be referenced as "accidentals", identified as a single/double (or, rarely, triple) sharpened or flattened version of a diatonic scale note, the selection of which depends on the pitch interval and key context. Commonly, a unique value is assigned to each pitch across all octaves.
- *Duration:* The period for which the note should sound. This is defined relative to the meter of the score, and the actual duration in time is dependent upon the pace of performance. Durations are commonly measured in fractions of a beat, with a particular number of beats occurring in each bar (for instance, 4/4 time, where each bar is composed of four quarter beats).
- *Intensity:* The volume of the note. In some representations, this may be dictated by the context in which the note occurs (for example, within a "forte" section of the piece), or a unique value within a fixed range may be assigned to indicate relative amplitude (such as the the *velocity* field of a MIDI file).

- *Accidental:* Where a non-chromatic scale is used, this attribute – composed of a maximum of 7 discrete values – indicates that the note is unaltered, or single/double/triple sharpened or flattened relative to its assigned pitch (see *Pitch* above). For simplicity, some representations treat accidentals as a trinary flag (unaltered, sharp or flat) since single accidentals are most common in classical notation.

- *Tie or Slur:* A note may be connected to another of the same pitch by a tie; this may be to extend the duration of the note but demonstrate its connection with a subsequent event. Alternatively, a slur may form a connection to a note of a different pitch, denoting that a curved transition should occur between the current a new pitch.

A rest may be considered as similar to a silent note, and associated only with a duration. The inclusion of rests in traditional scores allows notes to be written as offset from the start of each bar, eliminating the need for horizontal quantisation and allowing bars to be represented in a compact way. Rests are represented as objects in many containers, such as MusicXML or Sibelius scores, but some, such as MIDI, rely on absolute positioning of note objects using a *[bar, beat]* format.

In this study, musical data is extracted from Sibelius scores into a MusicXML format, where the values of the attributes listed here may be easily accessed and separated or processed at will. Western music is used exclusively, to simplify data extraction and allow for a direct comparison between the representation of each piece.

3.3 String Representations

The current thesis makes extensive use of digitally-stored musical scores, however the algorithms selected for evaluation are designed exclusively to process symbolic strings, such as linguistic or DNA data. To allow musical data to be passed to these algorithms as input, specific attributes are selected whose values can be represented as individual sequences. This section provides an overview of the attributes selected, and how they are prepared for processing.

As an example of the representation of written music as symbolic data sequences, the following pair of strings can be generated given the two-voice score in Figure 3.1:

A variety of representations relevant to the work detailed in later chapters is now presented.

- *Chromatic pitch:* MIDI note values $0 \leq v \leq 127$ as defined in the MIDI 1.0 standard (MIDI Association, 1996), each representing an individual semitone (12 notes per octave, as commonly found in Western music, where middle C = 60). To simplify pre-processing, rests are ignored, and tied notes & slurs are both represented as a pair of values, with their initial and subsequent pitch values stored individually and in sequence.

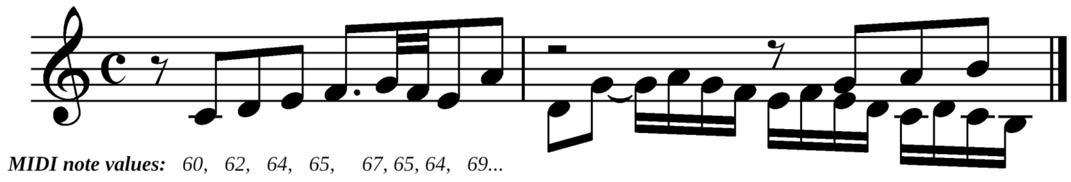


Figure 3.1: The first two bars from Bach's Fugue No. 1, WTC I., with MIDI note values shown for the first bar.

$p_{\mathrm{chr}^1} := [60, 62, 64, 65, 67, 65, 64, 69, 62, 67, 67, 69, 67, 65, 64, 65, 64, 62, 60, 62, 60, 59]$

$p_{\mathrm{chr}^2} := [67, 69, 71]$

For this study, tied or slurred notes are included as [*start*, *end*] pairs, and since the representation defines a twelve-note scale accidentals are not strictly required.

- *Diatonic pitch:* Note values $0 \leq v \leq 75$ taken directly from the *diatonic pitch* attributes of a Sibelius 7 representation of each piece, as defined in *Sibelius 7: Using the ManuScript language* (Avid Technology Inc., 2011) as "the number of the 'note name' to which this note corresponds, 7 per octave ...35 = middle C, 36 = D, 37 = E and so on". Rests are ignored, and accidentals are converted to their base notes.

  $p_{\mathrm{dia}^1} := [35, 36, 37, 38, 39, 38, 37, 40, 36, 39, 39, 40, 39, 38, 37, 38, 37, 36, 35, 36, 35, 34]$

  $p_{\mathrm{dia}^2} := [39, 40, 41]$

  The degrees present within a given diatonic scale are relative to a single key signature which is composed of a root note and a mode, the latter denoting the intervals between each degree. Diatonic pitch values are directly related to the chromatic notes they may represent, and so the chromatic surface is divided into $n(7/12)$ values, where n is the size of the set of all possible chromatic pitch values. This produces a mean of 12/7 chromatic pitches per diatonic value, and indication of which chromatic pitch is represented by any given value is given by the active key signature, or by the accidentals present, where these are sufficient to infer the current mode.

- *Chromatic / diatonic intervals:* For a string of pitch values p with $n = |p|$, an interval string d is generated, where $|d| = n - 1$, as follows:

$$d_i := p_{i+1} - p_i$$

  $d_{\mathrm{chr}^1} := [2, 2, 1, 2, -2, -1, 5, -7, 5, 0, 2, -2, -2, -1, 1, -1, -2, -2, 2, -2, -1]$

  $d_{\mathrm{chr}^2} := [2, 2]$

The interval representation allows for the existence of considerable transposition invariance between patterns which are played in differing keys. However, as shown by Cambouropoulos (Cambouropoulos, Crawford, & Iliopoulos, 2001), since each element of the interval sequence then represents the delta between two notes, if a contiguous portion of the string is divided into two segments then the same note will be represented by both intervals at the point of division. For example, consider the interval-based representation of p_{dia^2}. If it is separated into two groups, each containing the interval 2, then the portions of p_{dia^2} contained within each will be [39, 40] and [40, 41], thus the note with value 40 is present within both segments. It is important to be aware that operations such as this may not function as expected where adjacent groups are not separated by ≥ 1 symbol.

- *Note in chromatic / diatonic octave:* For a string of pitch values p, each element becomes

$$p_i := p_i \pmod{12}$$

$p_{\text{chr}^1} := [0, 2, 4, 5, 7, 5, 4, 9, 2, 7, 7, 9, 7, 5, 4, 5, 4, 2, 0, 2, 0, 11]$

$p_{\text{chr}^2} := [7, 9, 11]$

This representation is useful in the discovery of equivalent notes, or inverted chords, which occur in different octaves. For example, a C-major triad which is played in the third octave will contain the notes C_3, E_3, G_3, and the first inversion of this chord played in the fifth octave will contain the notes E_5, G_5, C_6. If the octave number is ignored, both patterns become equivalent since they contain the notes C, E, G.

- *Chromatic / diatonic contour:* For a string of pitch values p with $n = |p|$, a contour string c is generated, where $|c| = n - 1$, as follows:

$$c_i := \text{sgn}(c_{i+1} - c_i)$$

$c_{\text{chr}^1} := [1, 1, 1, 1, -1, -1, 1, -1, 1, 0, 1, -1, -1, -1, 1, -1, -1, -1, 1, -1, -1]$

$c_{\text{chr}^2} := [1, 1]$

This type of interval representation can be helpful in identifying patterns whose contours are equivalent – i.e. the amount by which two notes with a given numeric sign deviate is not significant, only that their signs are identical – such as where a phrase is repeated in a different musical mode, altering the magnitude of pitch intervals between notes.

- *Chromatic / diatonic histogram:* A score may be divided up into segments – fractions of a bar, for example – and a count kept of note occurrences for each segment. Where pitch values are chosen, each vector represents a pitch histogram from which it is possible to identify the most frequent, and potentially most significant, occurrences of pitch for that segment. It is of course necessary to pre-define the pitch value assigned to each element of the vector, so that a comparison may be drawn between them. This discussion relates to the direct use of pitch values, but pitch intervals, note durations, or other attributes may also be used.

For the excerpt shown in Figure 3.1, a vector of length 12 may be defined whose elements hold the count of chromatic pitch occurrences for the values 59-71. Splitting the example into groups of length 1 bar results in the following histograms:

$h_{\text{chr}} := ([0, 1, 0, 1, 0, 2, 2, 0, 1, 0, 1, 0, 0], [1, 2, 0, 3, 0, 2, 2, 0, 4, 0, 2, 0, 1])$

- *Duration:* Note duration values as defined in *Sibelius 7: Using the ManuScript language* (Avid Technology Inc., 2011), where 1 unit $= \frac{1}{256}$ of a crochet.

  $d_1 := [128, 128, 128, 192, 32, 32, 128, 128, 128, 128, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64]$

  $d_2 := [128, 128, 128]$

  Use of this representation alone can allow examination of only the rhythmic aspect of a musical score, without reference to the pitch being played, each note's placement, or the intensity with which it sounds.

- *Onset intervals:* From a string of note onset values, also defined in *Sibelius 7: Using the ManuScript language* (Avid Technology Inc., 2011) (where a bar has duration of 1024), a string of intervals may be generated in a manner similar to that of the pitch interval representation.

  $o_1 := [128, 128, 128, 192, 32, 32, 128, 128, 128, 128, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64]$

  $o_2 := [128, 128]$

In order to represent multiple voices within a score as a single symbolic sequence, some operation by which they may all be included is necessary. For the purposes of this thesis, where multiple notes found in an input score possess the same onset time, they may be converted to a sequence, and ordered by note value (for instance, an inversion of the chord C-E-G, with note values 60, 64, 55, will become [55, 60, 64]). Where multiple voices are present within a score, regardless of the polyphony of any given voice, they are treated individually and presented within the sequence as "separated" strings. This is achieved by their concatenation, separated by unique termination symbols to prevent an algorithm which seeks repeats from matching substrings across their boundaries. For instance, an input composed of the chromatic intervals of two voices may be generated as follows, with the symbol \$ chosen here as a unique terminator t_1:

$S := [d_{\text{chr}^1}, t_1, d_{\text{chr}^2}] := [2, 2, 1, 2, -2, -1, 5, -7, 5, 0, 2, -2, -2, -1, 1, -1, -2, -2, 2, -2, -1, \$, 2, 2]$

3.4 Representing Musical Data as Point Clouds

Given that all note attributes are required for a lossless model of a musical score, it may be considered an advantage for each element of a representation to include every attribute. In a representation composed of one symbolic string per attribute, a common index may be shared by each note in the score. However, this necessitates the access of several strings per note to extract the required elements. A representation where a single access retrieves all possible attributes avoids this behaviour.

Meredith et al. (2002) described such a representation during their presentation of the SIA family of algorithms. Multi-dimensional point clouds were chosen to represent a musical score, where each dimension referred to a specific note attribute, and each point within the space may therefore be referenced by a multi-dimensional vector representing a single note. If the number of dimensions is unrestricted but fixed, then all attributes can be modelled simultaneously, and so it is possible to contain all object data from a score within a single point cloud, resulting in no loss of information.

A disadvantage to this approach is that each note becomes unique unless it is an exact duplicate, whose existence could be considered an error. No equivalance naturally exists between patterns within the cloud if the vector describing each point is considered a symbol string. However, it is possible to select a set of n attributes a – where each attribute set forms a vector v_a of length n – which may then be translated given an n-dimensional vector v_t, and seek patterns within the cloud composed of p points which are equivalent under the translation $v_a + v_t$.

For example, consider the score segment in Figure 3.1. The following attributes might be assigned to each dimension:

$[onset, root, mode, pitch, duration]$

The first bar of the figure could then be represented with the following vectors:

$$[128, 60, 1, 60, 128] \tag{3.1}$$

$$[128, 60, 1, 62, 128] \tag{3.2}$$

$$[128, 60, 1, 64, 128] \tag{3.3}$$

$$[192, 60, 1, 65, 192] \tag{3.4}$$

$$[32, 60, 1, 67, 32] \tag{3.5}$$

$$[32, 60, 1, 65, 32] \tag{3.6}$$

$$[128, 60, 1, 64, 128] \tag{3.7}$$

$$[128, 60, 1, 69, 128] \tag{3.8}$$

In contrast to channel-based compression of symbol sequences, Meredith et al. (2002) presented the SIA family of algorithms. For this approach, events within the score were represented as points in multi-dimensional space, which enabled coupling of all related attributes to the event in question. This in turn allowed consideration of non-contiguous, cross-dimensional patterns, a geometic approach which seems intuitively of benefit in the analysis of music. The SIA family of algorithms described seek event groups in point space which may be translated to match other groups, forming sets of repeating patterns without pre-selected dominant attributes, whose events do not have to be adjacent, unlike channel-based string matching. This forms a powerful approach to repeat identification and piece segmentation, and some experiments within this thesis include a compression-based version of this algorithm, COSIATEC (Meredith, David, 2013), in their comparison. Given the strong results the SIA algorithms have returned over many musical studies, it is important to align the observed performance of the generalised compressors in these experiments against point-cloud based methods.

Meredith has identified key challenges in pattern discovery, along with drawbacks of the sequential representation and string matching approach, in a discussion of the SIA family (2006a). He began by citing the significance of repeats, before demonstrating the wide variation in structure and transform that must be considered, using examples from scores by Rachmaninoff, Barber and Bach. Highlighting the use of edit distance and separation of harmony into monophony, he showed how simple embellishment can break pattern equivalence, an issue which cannot be solved by allowing increased distance as ambiguity is likely to become dominant. A demonstration of the impracticality of inter-string matching to model harmony and voice transition was also given, showing the exponential growth in search space as each simultaneous note is encountered.

The geometric approach to pattern discovery can indeed overcome such difficulties. However, a single representation, or combination of representations, may not be ideal for every purpose. For example, use of diatonic intervals produces a more constrained search space, since a single step change may represent several chromatic pitches, resulting in a structure more sensitive to unit errors than its chromatic counterpart; such a representation might be useful for detection of change. Similarly, representing notes by chromatic pitch can provide greater similarity between pieces of a common key, a beneficial property when emphasis on primary pitch is desirable. Grammars by definition operate on sequences of symbols from the input's alphabet, necessitating the representation of music as strings for the purposes of this thesis. They are also trivial to construct, and limits imposed by their sequential

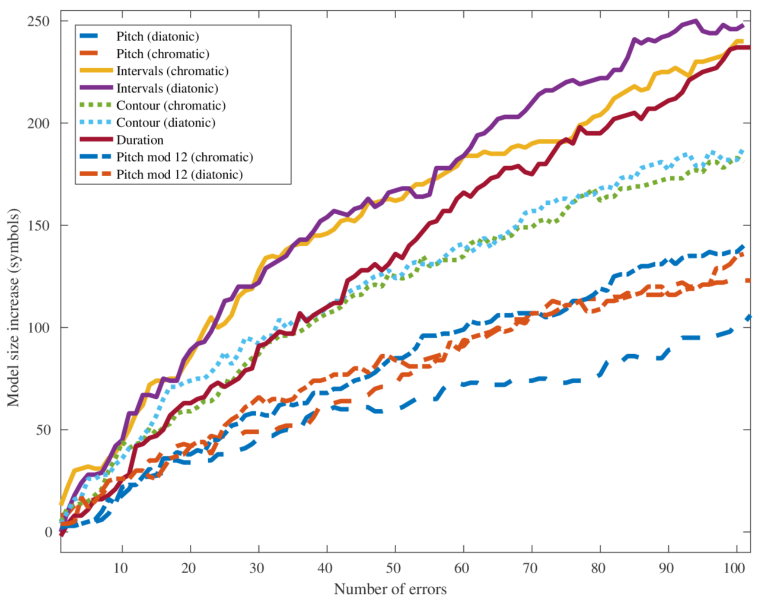


Figure 3.2: Reaction of grammar-based compressors to an increasing number of errors, in all available symbolic representations, for Bach's Fugue No. 10 from *Das Wohltemperierte Clavier* Book I. On average, diatonic intervals produce the strongest response.

nature serve to reduce search space complexity when seeking repeating patterns. Different models may be constructed from each note attribute, highlighting variations between them in underlying structure. This study investigates the performance of this simple representation on real-world data for various applications, demonstrating its continued relevance against more advanced techniques such as point sets.

Where a diatonic pitch representation is selected for a particular experiment, it is important to note that some ambiguity between pitch values is naturally introduced. The structure of this mapping is harmonic in nature and based on key scale positions, thus this representation incorporates some domain knowledge and may be seen as the result of pre-processing chromatic data. An examination of the score collection discussed below showed, on average, no direct mapping from diatonic to chromatic pitch existed for 13% of all diatonic values (with a standard deviation of 14%), with 4% and 22% of values being ambiguous in the best and worst cases respectively. This is a significant simplification of the input data, since the original chromatic values cannot be fully recovered from their corresponding diatonic pitches. However, the representation can provide enhanced results in suitable applications (Figure 3.2), as demonstrated in Chapter 5.

3.5 Multiple Viewpoint Systems

Witten and Conklin (1995) introduced *multiple viewpoints* as a distributed method of solving problems involving the processing of sequences, in response to the observation that existing context models were unable to properly represent complex event sequences, such as symbolic music, where intricate relationships exist between the types of event which occur within a domain. For example, a note's pitch, duration and onset point are separate values, but they are not chosen in an arbitrary and independent manner; within the music, a strong relationship is exhibited between them (Cook, 1994).

A viewpoint may be considered as belonging to one of three classes (Conklin, 2013b). A *basic viewpoint* can represent an attribute sequence from a discrete string of events, such as a sequence of pitch values occuring within a musical piece. A *derived viewpoint* may be seen as the result of processing a *basic* viewpoint, such as the output of a function which returns a sequence of pitch intervals, given a sequence of pitch values. Other examples of derived viewpoints include pitch or duration contour, where each element is a trinary value representing a neutral, positive or negative change from the preceeding input value. A *constructed viewpoint* forms a more powerful representation, where *basic* or *derived* data can be combined in a connected fashion – such as the product of inter-onset intervals (Pearce, 2018) and pitch intervals – allowing an algorithm which processes such a sequence to operate on a context formed from any number of recent events, and understood in a manner appropriate for the domain to which the input belongs. Viewpoints may be combined in an attempt to leverage the most useful features of each in any specific situation, using techniques such as arguments of the maxima or majority voting, to produce *multiple viewpoint* systems which explore many facets of each input's event sequences simultaneously. Multiple viewpoint systems have been shown to improve the accuracy of computational approaches to various musicological tasks, including music prediction (Conklin & Witten, 1995) and folk tune classification (Conklin, 2013b), when compared to basic context models, or algorithms which operate on basic viewpoints independently.

Given the success of multiple viewpoints, in particular with regard to symbolic music, it is reasonable to suggest their use may extend the ability of grammar-based models. Integration of multiple viewpoints into the grammar construction process is, however, not immediately trivial. A number of possible approaches exist to their inclusion, the most naïve being the construction of entirely separate models from each basic viewpoint, and the combining of results in a manner which aids the specific objective. As this study will show in Chapter 5, a weighted model combination can indeed prove useful for tasks such as the classification of folk tunes. However, a truly combined approach might be taken by generating a model which represents more than one viewpoint, and this may be problematic. A grammar-based compressor generates a model which exactly represents a single interpretation of its input; if several viewpoints are directly combined, the result may lead to an output structure which approximates all viewpoints, but does not entirely represent any as effectively as is possible if they are separately processed. Instead, multiple viewpoints may be presented to the compressor as concatenated, separated strings. In this case, the output is also likely to be formed as a concatenation of separate models, and any benefit to compression can only arise if the alphabets of different viewpoints contain common symbols, and similar patterns exist within them. This also offers no guarantees of being useful, since common symbols may have distinct meanings (such as the discrete value -1 in pitch interval and duration contour sequences), and so rules which match across viewpoints may not necessarily have any common contextual meaning. Viewpoints may be combined by constructing an input sequence for the compressor which is composed of elements which are multi-dimensional, but this is likely to strongly decrease the equivalence of substrings within the sequence since the number of unique elements may greatly increase, which does not aid compression. It is probable that such a representation will be of more use to an algorithm such as COSIATEC (Meredith, David, 2013) which is designed to work on multi-dimensional data. A more integrated approach would be necessary to effectively construct compact grammars from multiple viewpoints, and although a scheme of this type may prove successful, it is considered beyond the scope of the current thesis.

Nonetheless, this work does consider the use of *derived* viewpoints in the construction of compact grammars, alongside the weighting of models built from multiple viewpoints as described in Section 5.5. Many of the inputs fed into the compressors are composed of diatonic intervals, and pre-processed representations such as diatonic contour, and chromatic pitch modulo 12, are also investigated. For the purposes of this thesis, constructed viewpoints are not chosen, despite their usefulness in other studies. This decision was taken so that an evaluation of the performance of grammar-based compression on simple, individual viewpoints could be presented, as a foundation which may serve as a baseline in future studies which augment and extend the experiments presented here. Although simplistic grammars are arguably useful as models of musical structure, as discussed in Section 2.2, no large-scale study has yet been conducted to indicate their effectiveness on musicological tasks given a general population of musical scores. Such knowledge, in itself, a useful indication of whether it is possible to apply grammar-based compression to basic symbolic musical sequences – if a method exists which can equal the performance of existing work without requiring the use of complex or multiple viewpoints, it may provide a simpler approach to practical tasks, and, where this is not the case, the possibility of augmentation with a multiple-viewpoint approach remains, offering a strong potential for improved performance, as seen in tasks such as folk tune classification (Conklin, 2013b). As such, this study has chosen to focus generally on the use of single viewpoints, and consider only a limited set of derived viewpoints, as targets upon which the performance of grammar-based compressors may be quantified.

3.6 A gathered Corpus of Musical Scores

Ideally, the results of any study will represent the universe of all possible behaviours. That ideal is not always attainable, and as a substitute a set of generalisable behaviours may be examined, or a limited set of behaviours with specific significance to the problem. For this study, it is also desirable to conduct experiments on data which are representative of a large proportion of data available, and which are inclusive of a sufficiently large population of strongly structured compositions, with the aim of producing useful, generalisable results which represent more than just the sample under test. Historically important classical works were chosen as the basis for the study, since these possess clear structure – often with at least one analytical interpretation provided by an expert musicologist – and are available in a great enough quanitity to allow for a large number of experiments to be conducted.

Such an extensive dataset allows the establishment of a general relevance of results, and a cross-class evaluation of tested methods, given the wide variety of works and genres it may contain. A substantial collection of symbolic music data was gathered for this study, hereafter referred to as the *corpus*, and is composed of scores from the following sources: the *Acadia Early Music Archive* (Callon, 1998-2009); the *Choral Public Domain Library* (CPDL organisation, 2018); *Musopen*, a repository of free scores and recordings (Musopen organisation, 2018); *Music21*, a toolkit for computer-aided musicology with a large accompanying dataset (Cuthbert & Ariza, 2010); *KernScores*, an online symbolic music library (Sapp, 2005); a digital archive of the 1850 edition of *O'Neill's Music Of Ireland* (Chambers, 2015); the *Meertens Tune Collections*, a database of Dutch folk songs (Meertens Instituut, 2018); and the Johannes Kepler University *Patterns Development Database* (Johannes Kepler University, 2013), itself using data from *KernScores*. Some overlap between collections exists, for example between the *Music21* corpus and *KernScores*; for simplicity, duplicate pieces were not removed, and digital representations of multi-movement works were not split into their separate components, since such occurrences formed only a minority of the corpus. In total, 7961 digital scores were gathered, of which 7928 were converted to a suitable common format.

The MTC *Annotated Corpus v2.0.1* (van Kranenburg et al., 2016) is worthy of note, since each of its 360 members is bound to a "tune family" group as defined by musicologists at the Meertens Institute, and described by Volk and Kranenburg (2012) during their study of the relationship between musical features and similarity. This dataset is presented as "ground truth" which may be used to assess retrieval methods, and has formed the basis of several studies against which performance of such a method may be compared. 26 tune families exist within the *Annotated Corpus v2.0.1*, along

Source	Proportion
Music21	*37%*
KernScores	*28%*
O'Neill's Music of Ireland	*25%*
Meertens Tune Collections	4%
Miscellaneous	4%
Acadia Early Music Archive	1%
Choral Public Domain Library	1%
Musopen	<1%
JKU PDD	<1%

Table 3.1: Distribution of pieces by source; three primary sources make up the majority of the corpus.

with reference melodies (the "master" tune within the family), motif occurrences, phrase and song similiarities, and melody form within each strophe. In the present study, we use compression to classify each strophe by its tune family label.

Table 3.1 shows the distribution of pieces within the corpus by source.

Several score formats existed within the gathered data; compressed and uncompressed MusicXML (Good, 2001), Humdrum (Sapp, 2005), and ABC (Oppenheim et al., 2011). Scores not already in Sibelius 7 format (Avid Technology Inc., 2011) were converted, where, once loaded into the program and in an internal representation, data was exported to a custom XML format using a ManuScript plugin from the *sib2ly* utility (Sidorov, 2015). From this, the following information was extracted for each piece: pitch values and intervals (chromatic and diatonic), note duration, absolute position and accidental status (the latter represented within Sibelius 7 as trinary flags), and bar offsets. In addition, trinary contour vectors were constructed from both interval representations, where each interval was replaced by an element of $-1, 0, 1$, indicating the direction of change and thus representing the stepwise contour of the input. Note information was assigned to voices specified in the original digitised scores. At this stage, testing was conducted to ensure the extracted data was not corrupted in any manner, by converting each piece back into its original representation and comparing each corresponding event.

3.6.1 Distribution of Pieces by Source

The assembled corpus contains works by 39 known composers, with 3662 pieces whose composers are unknown. Of the latter, the majority belong to the three primary folk collections included: *O'Neill's Music Of Ireland (1850)*, the *Essen Folksong Collection* and *Ryan's Mammoth Collection*. The last

Composer	Proportion
Unknown	*46%*
Bach, Johann Sebastian	*20%*
da Palestrina, Giovanni Pierluigi	*17%*
Haydn, Joseph	4%
Corelli, Arcangelo	3%
Mozart, Wolfgang Amadeus	3%
Beethoven, Ludwig van	2%
Vivaldi, Antonio	2%
Others (<1% each)	4%

Table 3.2: Distribution of pieces by composer (to 0 d.p.); Bach and Palestrina make up a large portion of the corpus, but the majority of pieces have no recorded composer.

two of these are contained within the *KernScores* and *Music21* collections, the first consisting of approximately 6,255 folksongs originating from a variety of geographical regions, the second of 1,059 traditional Irish fiddle tunes. Table 3.2 provides details of the distribution of sources. Pieces are also categorised by period and genre; Tables 3.3 and 3.4 show how the collection is divided. For the majority of pieces, an accurate date of creation was not available; this "undefined" group forms the majority of the corpus, and is composed almost entirely of Folk music from the three listed collections. Period information for each piece was not independently sourced, instead metadata associated with a piece or collection was parsed for date information, and this recorded against each piece where available, with a label of "undefined" associated with any pieces for which a date could not automatically be found. Although creation dates are available for many folk tunes, this information was not included within the chosen collections, all of which were gathered in the 19th and 20th centuries but contain material from hundreds of years earlier. No attempt was made to verify the date information gathered, aside from manual checks as the scores from each collection were added to the corpus.

As an approximation of corpus complexity, times taken to construct grammars using ZZ were stored for each piece. These were used as a sort criterion to allow processing in order of run-time. Number of notes represented within each piece were also recorded. The relationship between symbolic length and approximate complexities for each piece is shown in Figure 3.3.

3.6.2 Correctness of Scores within the Corpus

The focus of this study is the response of compressors to a "real-world" collection of digital musical scores, whose data may be loosely qualified as being reasonably accurate with respect to the original

Period	Proportion
Undefined	*46%*
Baroque	*25%*
Renaissance	*18%*
Classical	7%
Romantic	2%
Medieval	1%
Georgian	1%
Tudor	<1%

Table 3.3: Distribution of pieces by period, the three largest being Baroque, Renaissance, and "undefined" – this last group contains pieces with a creation date which is either unknown, or was not recorded within the associated metadata.

Genre	Proportion
Classical	*55%*
Folk	*43%*
Undefined	2%
Jazz	1%

Table 3.4: Distribution of pieces by genre; most scores are either classical pieces or folk songs.

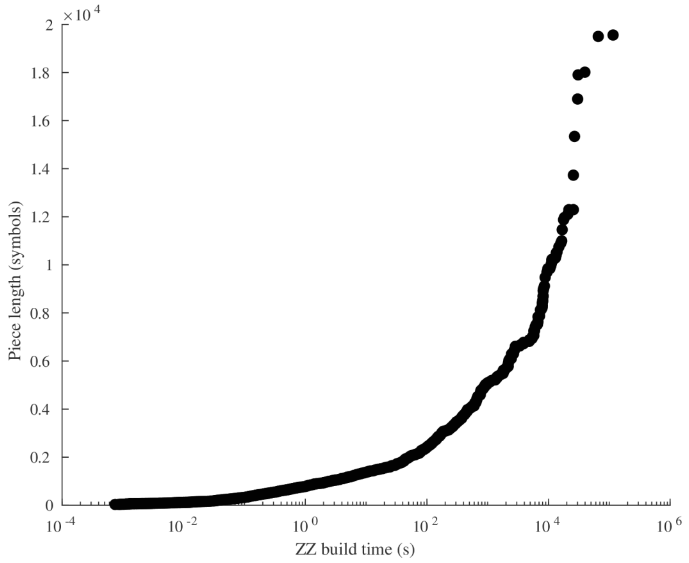


Figure 3.3: Relationship between symbolic length and approximate complexity of pieces within the corpus.

sources of transcription. The corpus of musical scores gathered for this study originate from a variety of sources, and are encoded in various formats, so it is reasonable to anticipate representational differences to exist after conversion to a common format. Given that the majority of sources are intended to aid academic study, it would be convenient to assume the bulk of the data were entirely accurate. However, this is unlikely to be true, especially where pieces are automatically transcribed or unverified against their original scoring. Indeed, some scores may contain errors prior to conversion. As such, data within the collection should only be considered partly accurate, and any observations made with this limitation in mind.

Some representational conventions which affect processing attempts by string-based algorithms were observed within the gathered corpus, in particular for pieces from Bach's *Das Wohltemperierte Clavier*, and it is possible that other issues exist generally. A good example in the case of the Bach pieces occurs because the score is represented by bass and treble clefs, assigned to the player's left and right hands respectively: when a pitch transition causes the notes of a motif to be split between the fingers of both hands, those notes are assigned to their related clef, and so neither clef contains the exact sequence of notes necessary to contiguously represent the pattern, making discovery of the instance by a sequential string-processing algorithm impossible.

The results of any experiments which assume entirely accurate data will obviously be affected by such inconsistencies. It would also be advantageous in some respects to equalise the distribution of composers, periods and genres present within the corpus, so that a more balanced and general dataset may be created. Nonetheless, the experiments contained within this study have been conducted on unmodified scores, with the aim of observing the results which may be expected given real-world digital music *collections*, including any inconsistencies and characteristics they may contain. Ideally, the music-processing algorithm will be robust to such variation, in the same manner as human perception. The accuracy of each converted output was tested to ensure the music data it contained matched exactly with that of the original input, but any further balancing or treatment of the collections is left for future investigation.

3.7 Summary

This chapter provided a general definition of the musical score in the context of the study's intentions, and discussed the various levels of abstraction, attributes and objects which a digital score contains. File formats which are commonly used to store digital symbolic music were also summarised, including proprietary encodings such as *Sibelius* (Avid Technology Inc., 2011), open-source encodings such as *MusicXML* (Good, 2001), *MIDI* (MIDI Association, 1996) and *MuseScore* (Watson, 2018), and text-based formats such as *ABC* (Oppenheim et al., 2011) and *Humdrum* (Sapp, 2005). It was noted that these formats may be used to store scores which have been converted from physical manuscripts, and that a large collection of 7961 such conversions, predominantly from the classical and folk genres, was gathered for the purposes of the experiments detailed in later chapters.

Having discussed the objects which are necessarily present within a digital score, the various string-based representations which have been chosen for the construction of experimental input data were defined, including chromatic and diatonic pitch & interval, pitch class as a function of scale degree within an octave, pitch interval contour, the segment-based pitch histogram, note duration and onset interval sequences. An example of the encoding of multiple sequences, via concatenation and separation with unique terminator symbols, was provided for a simple diatonic interval sequence consisting of two scored voices. These definitions show how a single symbol sequence which is suitable for input to a string processing algorithm may be produced from a polyphonic, multi-voice digital score, such as those present in the corpus, and demonstrate the pre-processing which occurs before the methods described later in this thesis are used to process score data.

The following chapter will discuss the production of straight-line grammars upon which this study is based, and define their properties, alongside introducing some other popular compressors selected as suitable baselines against which experimental results may be compared.

The grammar of a language can be viewed as a theory of the structure of this language.

Noam Chomsky, 1956

4 Straight-Line Grammars

THE research presented in this thesis focusses particularly on the application of grammars to the analysis of music, and specifically on the use of grammars as compressed models of digital score data, since this appears to be a promising technique (Sidorov et al., 2014) as discussed in Chapter 2. Such models appear capable of identifying structure which is musicologically relevant to a score (Nevill-Manning & Witten, 1997), and responding to changes in structure in a manner which suggests a degree of recognition of musical context (Sidorov et al., 2014). This chapter provides a definition of grammars, and their important properties, as theoretical structures which are central to this study. Other compression techniques, which are used to produce a baseline against which experimental results may be compared, are also described. Although the definition of a grammar does not place it within the musical domain, a grammar may indeed form a lossless model of a musical sequence, and so its inclusion is important in laying the foundation upon which the contributions in this thesis – an exploration of grammars with respect to music analyis, augmentation of grammars with flexible production rules, and approximation of constituent gain as a heuristic to reduce construction time – are built.

A straight-line grammar is a formal, hierarchical model which generates exactly one output string. Existing work has shown that grammars may represent a descriptive model (Mondol, 2020) in the context of Kolmogorov Complexity (Kolmogorov, 1963). Given that music may be represented as a symbolic string s, its Kolmogorov Complexity can be defined as follows:

$$K_T(s) = \min\left(|P|, T(P) = s\right)$$

Where T is a Universal Turing Machine (Turing, 1936), and P is a program which exactly generates s. $K_T(s)$ cannot be exactly computed (Chaitin, 1966), but it may be approximated as an upper semi-computable function (Li & Vitányi, 2008) (defined as a partial function $f \colon \mathbb{Q} \to \mathbb{R}$ which, if there exists a computable function $f(x, k) : \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$, where k is the level of approximation chosen in calculating the parameter of $f(x)$, may be approximated from above). As such, minimisation of $|P|$ is equivalent to approximating the Kolmogorov Complexity of s. Minimisation of grammar size $|G(s)|$ produces a small encoding which may exactly reproduce s using a simplistic expansion process, analogous to T. As stated in (Carrascosa et al., 2010), this approach allows computation of a variation of $K_T(s)$.

4.1 Straight-Line Grammars

This section defines the components which constitute a straight-line grammar, as used throughout this dissertation. Such definition is important to ensure the work may be aligned with existing studies, and follows accepted conventions from the literature (Carrascosa et al., 2011) for consistency. It is from these components that an encoding is generated, such as that which is output from a grammar construction algorithm such as ZZ (Carrascosa et al., 2011); knowledge of these components is key to understanding the grammar construction process, and how an encoding may be generated, encoded and quantified from a given model.

For this study, the following definition has been adopted. As shown by Carrascosa et al. (2011), a straight-line grammar G may be defined by the following components:

$$G = \langle \Sigma, N, P, S \rangle$$

Σ is the set of *terminal symbols*, and comprises the finite symbolic alphabet from which the input string s is formed. N is the set of unique *non-terminal symbols*, where Σ and N are disjoint. P contains the grammar's *production rules*, each of the form $A \to \alpha$, where $A \in N$, and α is a sequence from $(\Sigma \cup N)^*$. S is known as the *start symbol*, and refers to a special production rule from P, with $S \in N$, known as the *primary* rule (Charikar et al., 2005); all other rules in P are referred to as *secondary*. Expansion of the grammar begins with S, and whenever a non-terminal $A \in N$ is encountered it is replaced with α from the right-hand side of the corresponding production rule..

G is encoded as a string, composed of a sequence of symbols, each an instance of $g \in (\Sigma \cup N)$, representing a concatenation of all rules in P in the form $(P_S, P_1, P_2, \ldots, P_n)$. Rules are separated by a special terminator symbol $|$, making it unnecessary to include $A \rightarrow \alpha$ explicitly within the encoding; instead, each rule is encoded as $\alpha|$, with the order of their concatenation defining their position within P, and, consequently, their reference $A \in N$. This results in a string of the form $\alpha_S|\alpha_1|\alpha_2| \ldots \alpha_n|$. Conventionally, the size of G may be calculated as follows:

$$|G| = \sum\nolimits_{A \rightarrow \alpha \in P}(|\alpha| + 1) \tag{4.1}$$

G must be fully expanded in order to reproduce s. The expansion of a rule $r \in P$, with non-terminal $A_r \in N$ is denoted by $\langle A_r \rangle$, and overall expansion of G always begins with $\langle S \rangle$, continuing iteratively until $c \notin N$ for all $c \in \langle S \rangle$. G is therefore acyclic, and there exists a global expansion function $\varepsilon(G) = \langle S \rangle$ for all such grammars.

4.1.1 Grammar-based Compression

The goal of a smallest grammar algorithm is to minimise $|G|$ by generating a grammar forming the *smallest possible* encoding of the input string s, resulting in $|G| < |s|$. This is approached by only including production rules whose symbols $A \in N$ will replace instances of substrings in s in a combination which produces the smallest possible encoding *overall*. Given a straight-line grammar G which exactly reproduces the string s, where $|G| < |s|$ the encoding represents a compressed encoding of s, although it may not necessarily be the smallest encoding possible.

During the construction of G as a compressed encoding of s, a rule r should not be included in P unless it results in compression. Given $v = \langle A_r \rangle$, replacing n instances of that substring v in s with the symbol $A_r \in N$, it must hold that $|\alpha_r| + 1 < |v| \cdot n$ in order for r to remain compressive within G. This inequality assumes that no other rules are formed from substrings which are also substrings of v; where this is the case, $|v|$ is reduced, and r becomes less compressive.

Computationally, construction of a smallest straight-line grammar (one which reproduces only its input string) is a complex process proven to be NP-Hard (Lehman & Shelat, 2002). However, a close approximation may be discovered with lesser complexity, making algorithms which generate a compact grammar, such as ZZ, suitable for practical use. Existing grammar construction methods vary in complexity and performance. Carrascosa et al. (2010; 2011; 2012) showed the time complexity of IRR schemes to be $O(n^2 \, log \, n)$, compared to $O(n^7)$ for ZZ, where n is the length of the input in symbols. It is important to note that this represents maximum complexity; in practice, convergence usually occurs far earlier.

When drawing comparisons in performance between multiple compression techniques, the techniques chosen should bear sufficient similarity in function and output to enable a relevant comparison to be made, and in an equivalent context. This study selects ZZ and IRR-MC as well-known algorithms which may be suitably compared, and as a basis for further development. IRR-MC is included so that experimental results obtained from a less powerful compressor may be compared to those obtained using ZZ, to explore the hypothesis that there is a relationship between compressor strength and performance for particular tasks. The process of ZZ optimisation can also be compared to methods this study presents, allowing them to be evaluated against current knowledge.

4.1.2 General Grammar Construction

A grammar may be constructed based on the definition in Section 4.1, as follows. Initially, P is empty and N contains only a reference to S. The right-hand side of the rule associated with the start symbol S is initialised with the input string s, producing $S \rightarrow s$, and thus the unique set of its symbols are added to Σ. Substrings of $\langle S \rangle$ are then added as new production rules and their occurrences in P replaced with a rule reference symbol, which itself becomes a member of N. How and when these replacements are made depends on the grammar construction method. The goal of a smallest grammar algorithm is to minimise $|G|$ by only generating new production rules which replace substrings in P with their respective references in a combination which gives the greatest overall reduction in grammar size.

4.1.3 IRR-MC and ZZ in Operation

IRR-MC begins with P empty and N containing only S, as a reference to the primary rule within P. Variable S is initialised with the input string s, and all members of the unique set of its symbols are added to Σ. At each iteration, the most compressive substring within the right-hand sides of all rules in P is added as a new production rule, and its occurrences replaced with a rule reference symbol A_n, which itself becomes a member of N. As shown by Carrascosa et al. (2011), replacement of a substring ω which occurs n times in the right-hand sides of the rules in P with a new rule causes a decrease in $|G|$ of $(|\omega|-1) \cdot n$, and an increase in $|G|$ of $|\omega|+1$. Assuming a function $f(G, \omega, n) = (|\omega|-1) \cdot n - (|\omega|+1)$, the *most compressive* substring may then be defined as $arg\,max\,f(G, \omega_i, n_i)$ for all ω in the right-hand sides of all rules in G. The process is complete when no ω exists which further reduces $|G|$, and the result is a greedy minimisation of $|G|$. The grammar produced will exactly generate s alone upon expansion.

In contrast, ZZ traverses a lattice of possible substrings, known as constituents, where moving to a different node within the lattice represents addition or removal of a constituent from the current set, and so of a production rule from P. Traversal ends when no move to a neighbouring node results in the possibility of constructing a more compact grammar from that node's constituent set, thus locally minimising $|G|$.

4.1.4 Approximation Ratio

Grammar construction for a string s containing c constituents possesses a combinatorial space of $O(2^c)$, making selection of a smallest grammar for a large s untractable. Instead, an approximation is commonly sought, in order to allow the production of an upper semicomputable function (as defined at the beginning of the chapter). Given a grammar construction algorithm $\lambda(s) \mapsto G(s)$, and a perfect smallest grammar construction algorithm $\Lambda(s) \mapsto G(s)$, an approximation ratio for the smallest grammar problem (Charikar et al., 2005) can be defined as a function $f(n)$:

$$f(n) = \max_{s \in \sum^n} \left(\frac{|\lambda(s)|}{|\Lambda(s)|} \right)$$

For a specific class of grammar constructors, *global algorithms* (2005), Charikar et al. defined the following properties for any grammar G generated, where g is the concatenation of all the right-hand sides of rules in P:

- For any pair of right-hand side rule definitions$(\alpha, \beta) \in P$ with $|\alpha| \geq 2$ and $|\beta| \geq 2$, $\alpha = \beta \iff \langle\alpha\rangle = \langle\beta\rangle$.
- For all non-overlapping symbol pairs $(\alpha, \beta), (\beta, \gamma) \in g$, $(\alpha, \beta) \neq (\beta, \gamma)$.
- For each $A \in N$, the count of all instances of A occurring in g is ≥ 2.

Given these properties, the approximation ratio of this class of algorithm may be defined:

$$O(\left(\frac{n}{\log n}\right)^{\frac{2}{3}})$$

The same study provided a proof (*Lemma 1*) demonstrating that for a smallest grammar $G(s)$ of this class, with $n = |s|$, encoding size may be taken as follows:

$$|G(s)| = \Omega(\log n)$$

4.1.5 Performance of Grammar Construction Algorithms

Carrascosa et al. (2010; 2011; 2012) showed that a variety of existing grammatical compressors performed identical steps during the construction process, differing only in constituent score function, and presented an algorithm unifying these approaches. At each iteration of the algorithm, repeating terms are identified within S and P, and the reduction in $|G|$ resulting from their replacement by reference to an additional rule in P is calculated. The term t providing the greatest anticipated reduction (over n instances) is chosen as the next constituent, replacement is made, and iteration continues. The algorithm terminates when no further reduction is possible. The study defined three principal score functions – Maximal Length (ML, $\max(|t_i|)$), Most Frequent (MF, $\max(n_i)$), and Most Compressive (MC, $\max(|t_i|n_i)$) – and demonstrated that all IRR schemes are prone to loss of compression due to greedy constituent choice blocking globally more compressive combinations at subsequent iterations. In answer to this, Carrascosa et al. described *Occurrence Optimisation*, and presented a search space traversal method, ZZ, capable of removing sub-optimal constituents and arriving at a local minimisation of $|G|$ without the complexity associated with an exhaustive examination of constituent combinations.

Existing grammar construction methods vary in complexity and performance. The Sequitur algorithm's smallest grammar (Nevill-Manning & Witten, 1997) is bounded $O(log\, n)$, and the LZ77-based AVL-grammar algorithm (Rytter, 2002) can approximate a ratio of $O(log\, n)$ in $O(n\, log\, |\Sigma|)$ time, where Σ is the input string's unique symbols. Charikar et al. (2005) presented an approach capable of ratio $O(log\, n/m)$, where m is the size of the smallest grammar per input. Carrascosa et al. (2010; 2011; 2012) showed time complexity of IRR schemes to be $O(n^2\, log\, n)$, compared to $O(n^7)$ for the more powerful ZZ. Consistently smaller models were produced by ZZ relative to IRR-M*x* across inputs of varying size. Benz and Kötzing's GA-MMAS (2013) offered slightly smaller models than ZZ, at the cost of greater time complexity. Due to the unavailability of a GA-MMAS implementation, ZZ's minimal time complexity, and the volume of data to be processed, we select the latter as the principal compressor for this study. We also include IRR-MC, as the optimum IRR-M*x* scheme, to provide a direct comparison against less compressive models in our experiments.

4.1.6 Hierarchical Structure

The hierarchical structure of a straight-line grammar has useful properties when modelling many different data types, including musical data. An example of this may be seen in a later chapter; Figures 5.3 and 5.4 provide an example of the manner in which structure may be represented for musical compositions. Any production rule may contain references to sub-rules in P, making grammars suitable for modelling sequences which exhibit nested structure when decomposed into significant segments. Music is known to be composed of units such as melodic phrases which are often present within recurring passages, and although it is reasonable to state that disjunctive segmentation alone such as that presented by a straight-line grammar cannot entirely explain a score (Singer, 2004), it is still possible that such disjunctive segmentation in *part* represents structure intentionally defined by the composer. As such, nested rules within the grammar remain useful to segmentation-based tasks (Sidorov et al., 2014).

Grammar-based compression is of particular interest for this study, not only due to the promising results of previous work, but because such a grammar models a hierarchical, approximate smallest encoding of some particular input data, without loss. Top-level rules may resemble semantic segmentation as defined by humans, with low-level rules forming the building blocks from which higher level structure is formed. Alphabet symbols may be directly edited, allowing simultaneous processing of multiple sequence positions, across multiple sequences. This study begins by proposing that a compressive grammar for a given piece can effectively model such hierarchy, and is therefore useful in performing musical tasks which are based upon segmentation.

4.2 Compressors selected for Study

This study adopts ZZ as its primary existing compression method, since one of its key aims is to examine whether hierarchical, compressed models may be directly leveraged in analytical tasks. As discussed earlier, IRR-MC is included so that direct comparison against a less optimised compressor may be made. The geometric point-set compressor COSIATEC (Meredith, 2006a) is known to be a more powerful pattern discovery method on symbolic music data. However, it is not designed to perform optimally on many of the tasks selected for testing, and produces a very different encoding to that of ZZ or IRR, making a direct comparison between the output of COSIATEC and a grammar-based compressor unfeasible. Where this output is used as part of a subsequent processing task, encoding differences may also cause a variation in task performance which is not immediately reflective of the ability of the compressor, making the comparison of results between compressor types less meaningful. Most importantly, as a point-set algorithm specifically designed to process polyphonic music, COSIATEC accepts an input of an entirely different form, distinct from the strict sequences of string-based compressors such as ZZ or IRR. When seeking repetition, COSIATEC is able to build patterns from separate events spread across the temporal domain; using the representation style described in Section 3.3, a string-based compressor can only group events which are adjacent within a given time sequence when selecting a pattern. Such differences are likely to cause the results of the experiments conducted for this study to vary widely between compressor types, without necessarily providing a useful foundation for the analysis of inter-class performance on tasks specifically designed for string-based compressors. Due to these factors, COSIATEC is not selected for inclusion in these experiments.

Different compression techniques are notably capable of producing useful and interesting results given musical data. Many previous studies have employed popular file compressors for classification and segmentation. Cilibrasi, Vitányi, and Wolf (2004) experimented with different methods, including GZIP (Deutsch, 1996), BZIP2 (Seward, 1996) and the Linux standard command *compress* (Welch, 1984), during their investigation of clustering by normalised information distance. Li, Chen, Li, Ma, and Vitányi (2004) also use GZIP in their presentation of a normalised information distance. Louboutin and Meredith (2016) considered the Lempel-Ziv Welch algorithm for classification and pattern discovery, but stated that they chose LZ78 (Ziv & Lempel, 1978) as this would require storage of an unbounded alphabet within the compressed output. Since input alphabets are constrained for symbolic music data, this study includes LZW (Welch, 1984) within its set of compressors, along with GZIP, and run-length encoded (Tsukiyama et al., 1986) Burrows Wheeler Transform (Burrows & Wheeler, 1994), the last of which forms core layers in the BZIP2 compression stack. These referenced studies demonstrated the effectiveness of generalised compression when applied to symbolic music data, and it is useful to include this class of algorithm so that experimental results may be placed within a wider context.

Although model size is the primary metric in the experiments within this study, it is not used to directly compare compressors due to the differences in encoding. An additional $c = |P|$ separator symbols are required for grammars, with rules being explicitly stored. LZW requires $l = |\sigma|$ additional symbols, where σ is the set of symbols present in the input, since this input alphabet forms an initial dictionary. However the remainder of the dictionary is generated upon decode, and so is not explicitly encoded. COSIATEC simply returns the total number of points and offsets within the patterns it discovers, defined as *Maximal Translational Equivalence Classes* (Collins & Meredith, 2013), along with a set of residual points not referenced in any MTEC. Where GZIP is used, the literal file length in bytes is taken as the size of the compressed model. It is clear that encoding size alone cannot be used for cross-method evaluation, and so compression ratios are included in the presented experiments where appropriate. It is worth noting that differences in encoding may obviously affect measurement of compressor sensitivity, even where the process by which a compressor is applied to a particular task remains identical.

4.3 Examples of Grammar-based Compression

This section contains discussion and simple examples of how a grammar may be constructed using different schemes, all of which are relevant to the work presented in this thesis: *Iterative Repeat Replacement* using a *Most Compressive* score function, *Iterative Repeat Choice* with *Occurrence Optimi-*

sation, and *Zig-Zag*. For the latter, the search space of constituents associated with the algorithm – the *Lattice of constituents* – is briefly discussed, and work presented in later chapters may be understood in reference to this structure. For each example, an encoding of the grammar will be used as its representation, with unique terminator symbols $\$_1, \$_2, \ldots, \$_n$ used to separate S from each production rule, and production rules in P from each other. Non-terminals from N are defined as $r_1, r_2, \ldots, n$, where r_n is the n^{th} production rule in P.

4.4 Iterative Repeat Replacement

Carrascosa et al. (2011) define a general scheme which most offline grammar construction algorithms may be shown to follow. Beginning with assignation of the inputs string's symbolic sequence s to S, construction of the grammar follows an iterative procedure which continues until no reduction in the grammar's length, as defined in Section 4.1, is possible. At each iteration, a score function is used to evaluate the "best" substring to replace within S or P, and replacement is carried out by addition of a new production rule to P, assigned its own, unique non-terminal symbol which is added to N. This non-terminal symbol is subsituted for all non-overlapping occurrences of the substring, chosen greedily from ascending indices of the grammar's sequence, except for the sequence on the right-hand side of the new production rule. In this way, a grammar G is generated which represents a compressed encoding of the original input s.

Given a simple input string $s = abcabcdbcebc$, a brief example of grammar construction can be outlined. First, s is added to the grammar, resulting in the encoding $a, b, c, a, b, c, d, b, c, e, b, c\$_1$, with length $|G| = 13$. Assuming *Most Compressive* is chosen as a score function, the score of each substring in the encoding will be $|t_i|n_i$ as defined in Section 4.1.5. The set of all possible substrings of length ≥ 2 and with number of occurrences ≥ 2 are $\{ab, bc, abc\}$. These are of length $[2, 2, 3]$, and occur $[2, 4, 2]$ times within the input s, producing MC scores of $[4, 8, 6]$.

If the substrings are ordered firstly by descending score, and secondly by their index, the substring bc is chosen first for addition to the grammar. Knowing that $n = 4$ occurrences of the substring of length $l = 2$ will be replaced, and each occurrence will be substituted by exactly one non-terminal to represent the new production rule, the gain which the replacement alone will offer is $n(l - 1) = 4$. However, the production rule itself must be encoded, and terminated by an additional, unique symbol. This results in a loss of $l + 1 = 3$. Therefore, the overall gain is $n(l - 1) - l + 1 = 1$. The existing grammar is of length 13, and so choosing bc as a new production rule would result in $|G| = 12$. Since this is less than the length of the previous encoding, bc is selected for addition to the grammar. Replacement then occurs in a greedy fashion over all indices of the encoding, here producing

$a, r_1, a, r_1, d, r_1, e, r_1, \$_1, b, c, \$_2$, which represents $S = a, r_1, a, r_1, d, r_1, e, r_1$, and $P = [b, c]$, where $|G| = 12$. At this point, a new iteration begins.

Within this new encoding, the set of all possible substrings is $\{[a, r_1]\}$. No other repeats now exist. The substring is of length 2 and occurs twice within the encoding, resulting in a score of 4. Knowing here that $n = 2$, $l = 2$, and so $n(l - 1) - l + 1 = -1$, it follows that $|G| = 13$ for a grammar containing a, r_1 as a production rule. Since this is greater or equal to the length of the previous encoding, it is not retained, and G remains the same as in the previous iteration. Had a, r_1 been included in G, it would have resulted in the encoding $r_2, r_2, d, r_1, e, r_1, \$_1, b, c, \$_2, a, r_1, \$_3$ of length $|G| = 13$. As the most compressive substring offered no reduction to $|G|$, the algorithm now terminates, and outputs the following, final encoding:

$a, r_1, a, r_1, d, r_1, e, r_1, \$_1, b, c, \$_2$

4.5 Occurrence Optimisation

Considering the process described above, no provision is made for *optimisation* of the occurrences of substrings which are replaced within S or P, as new production rules are iteratively added. If a grammar G contains a production rule r_1 which replaces substrings matching w_1 in S or P, a new candidate rule r_2 may refer to substrings w_2 in the original input s which *overlap* with instances of w_1. It is entirely possible that r_2 may offer sufficient gain to the grammar to be useful in reducing $|G|$, but only if one or more instances of w_1 are not replaced by the non-terminal of r_1. Indeed, r_1 may continue to offer sufficient gain even if those instances are *not* replaced by its non-terminal in the encoding, but since all occurrences of w_1 were replaced in the process of IRR, evaluation of the score for the candidate r_2 cannot accurately reflect the potential for gain provided by r_1, r_2 in combination, and the candidate r_2 may not be chosen during that iteration, or perhaps at all, for inclusion in G. This may result in a sub-optimal encoding, with a length $|G|$ greater than is possible if the *occurences* of the substrings each production rule replaces during grammar construction were optimised.

Carrascosa et al. (2011) introduced the process of *Occurrence Optimisation*, in which all encodings in S and P are represented by Directed Acyclic Graphs (Thulasiraman & Swamy, 2011), where each edge between two neighbouring nodes represents a symbol in the rule's sequence, and edges between non-neighbouring nodes represent substring occurrences which may be replaced by production rules present within G. In this manner, the choice of constituents from which to form production rules is separated from the choice of substring occurrences in the encoding, and the optimal model which may be produced given a specific set of production rules may be found by performing a minimal parse of all graphs. This is termed *Minimal Grammar Parsing*, or *MGP*.

MGP may be performed by applying any algorithm which is capable of finding a shortest path through a Directed Acyclic Graph, such as Dijkstra's algorithm (Dijkstra, 1959) or the Bellman-Ford algorithm, first proposed by Shimbel (1954), to a graph representing the input string and all substrings forming its constituents. Tracing the shortest path through this graph results in a sequence of symbols: terminal symbols results from the traversal of an edge containing an element present in the original input, and non-terminal symbols results from the use of an edge which is associated with an available constituent. The sequence of symbols produced during the traversal forms the grammar's encoding, and represents the smallest encoding it is possible to generate given the available constituents. The computational complexity of MGP depends on the shortest path algorithm used. For the purposes of this thesis, it is $O(|E| + |V|)$ where E is the set of all edges and V the set of all vertices, since every edge which leads to any vertex $v \in V$ must be considered in order to ascertain which is the last link in the shortest path arriving at v, this step must be performed for all vertices in the graph, and each edge arrives at exactly one vertex.

In an extension of IRR which is able to optimise substring occurrences, called *Iterative Repeat Choice with Occurrence Optimisation*, or *IRC-OO* (Carrascosa et al., 2011), the score function is replaced by a Minimal Grammar Parse of S and P, resulting in scores which are optimal with respect to the new candidate constituent, unlike *Most Compressive*. This allows the actual gain which a candidate can provide to a given grammar G to be known, ensuring that new production rules which are capable of reducing $|G|$ are not ignored due to the failure of the score function to observe optimal encodings during evaluation. IRC-OO was shown to allow more compact grammars to be discovered for certain inputs (Carrascosa et al., 2011).

4.6 Lattice of Constituents

The work of Carrascosa et al. (2011) also defined a search space of all possible candidate constituents, each representing a set of substrings which may be replaced through the addition of a production rule to a given grammar. This space represents all possible combinations of production rules, including the empty set, and since a globally optimal model G contains ≥ 0 constituents, the space must contain at least one combination which, when used in a Minimal Grammar Parse, generates the globally optimal encoding. An example lattice may be considered for the input string $s = abcabcdbcebc$, where the set of all repeating substrings is $\{ab, bc, abc\}$. If these are considered constituents 1, 2 and 3, the search space of all possible combinations can be visually represented as shown in Figure 4.1.

For any node within the lattice, it is possible to traverse at least one edge to arrive at a different node. Each edge represents the addition or removal of a constituent from the current combination:

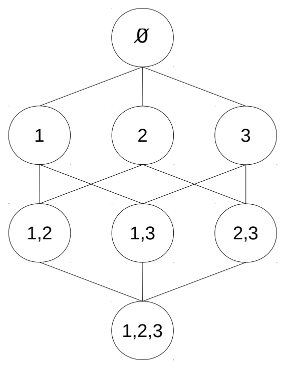


Figure 4.1: An example of a Lattice of all possible constituents – i.e. all potential production rules – for a grammar whose input contains exactly 3 repeating substrings.

where the edge leads downwards, a new constituent is added to the set, and where it leads upwards a constituent is removed. Motion from one node to another represents a single operation, and motion from one level to another represents a change in the number of constituents within the combination. For n constituents, the maximum depth of the lattice is $n + 1$, and there will be a total of 2^n nodes within it. The maximum width of the lattice is defined as:

$$\frac{n!}{\lceil \frac{n}{2} \rceil!(n - \lceil \frac{n}{2} \rceil)!}$$

Grammar construction algorithms may be said to begin at the top-most node, containing an empty set, and aim to traverse the space to arrive at the node containing the globally-optimum combination of constituents. Performing MGP using the set of constituents any given node represents results in production of a grammar G, for which an encoding length $|G|$ can be calculated. Optimal nodes in the lattice are those for which $|G|$ is globally minimal.

4.7 ZZ

In the same work, Carrascosa et al. presented the optimisation method Zig-Zag (ZZ) (Carrascosa et al., 2011), designed to traverse the lattice in search of the first node which resulted in a locally-optimum value for $|G|$. Although any node may be selected as a combination of constituents C to begin with, it is common to begin with $C = \emptyset$. ZZ operates in two phases – *top-down* and *bottom-up* – which are identical except for the vertical direction they explore within the lattice, examining connected nodes

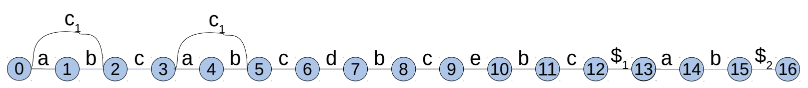


Figure 4.2: An example parse graph for the input *abcabcdbcebc*, and the constituent *ab*. The minimum encoding length is 14, and may be found by Minimal Grammar Parsing.

in the level above or below respectively. The goal of both phases is to discover the encoding lengths associated with each connected node, each representing a possible choice of constituent to add / remove from the current set, so that a node which minimises $|G|$ to smaller value than the current node can be chosen. Such a choice results in the progressive optimisation of G with respect to encoding length. Where no candidate node produces a smaller value for $|G|$, no movement occurs, and the phase changes so that results of the opposite operation can also be evaluated. When movement in either direction cannot reduce $|G|$, the algorithm terminates, and returns the selected set of locally-optimal constituents. Ideally, this set will also represent the global optimum, but this will not be true if the global minimum value for $|G|$ has not been observed by the algorithm for any nodes encountered during traversal.

Beginning with the input string $s = abcabcdbcebc$, and the set of all possible substrings $\{ab, bc, abc\}$ assigned to constituent numbers [1, 2, 3], a simple example of grammar construction using ZZ can be outlined. Initially, since the set of chosen constituents is empty, $C = \emptyset$. ZZ begins in the *top-down* phase, so the effect of adding a single constituent to C can be explored. Since the encoding length of a string s is $|s| + 1$, as a terminator symbol must be appended to mark its end, it is known that $|G|$ for $C = \emptyset$ here is 13. For a combination to be selected as an improvement on $C = \emptyset$, it must result in $|G| < 13$.

The set of nodes which must be explored from the node $\emptyset$ is $[\{1\}, \{2\}, \{3\}]$. The parse graph for the first node, representing the substring *ab* is shown in Figure 4.2. By parsing the graph starting from the first node, of index 0, such that the minimum number of vertices are visited, the encoding $r_1, c, r_1, c, d, b, c, e, b, c, \$_1, a, b, \$_2$ of length 14 is produced.

Next, the parse graph for the second node, representing the substring *bc*, is shown in Figure 4.3. By again parsing the graph starting from the start node such that the minimum number of vertices are visited, the encoding $a, r_1, a, r_1, d, r_1, e, r_1, \$_1, b, b, \$_2$ of length 12 is generated.

Finally in this phase, the parse graph for the third node in the current level of the constituent lattice, representing the substring *abc*, is shown in Figure 4.4. Again, performing a minimal parse of

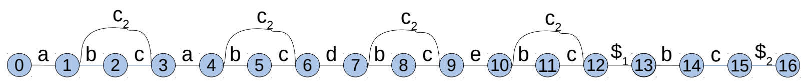

0 a 1 b 2 c 3 a 4 b 5 c 6 d 7 b 8 c 9 e 10 b 11 c 12 $1 13 b 14 c 15 $2 16
c2
c2
c2
c2

Figure 4.3: An example parse graph for the input *abcabcdbcebc*, and the constituent *bc*. The minimum encoding length is 12, and may be found by Minimal Grammar Parsing.

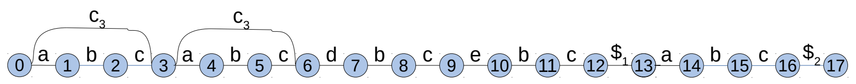

0 a 1 b 2 c 3 a 4 b 5 c 6 d 7 b 8 c 9 e 10 b 11 c 12 $1 13 a 14 b 15 c 16 $2 17
c3
c3

Figure 4.4: An example parse graph for the input *abcabcdbcebc*, and the constituent *abc*. The minimum encoding length is 13, and may be found by Minimal Grammar Parsing.

the graph produces the smallest possible encoding for this constituent combination, resulting in the encoding $r_1, r_1, d, b, c, e, b, c, \$_1, a, b, c, \$_2$ of length 13. Now that all three encoding lengths are known – [14, 12, 13] – it is possible to select the first occurrence of the minimum, and to verify it is indeed less than the smallest known encoding at this point. At 12, it is smaller than the encoding for $C = \emptyset$ which is 13, and so the set of constituents changes to $C = \{c_2\}$ as ZZ moves to the corresponding node.

The next set of nodes reachable from the new position are $\{[1, 2], [2, 3]\}$. Figure 4.5 shows the parse graph for the constituent combination $\{1, 2\}$. Two encodings of identical length are possible for this graph, and selection of the first of these results in $r_2, c, r_2, c, d, r_1, e, r_1, \$_1, b, c, \$_2, a, b, \$_3$ with length 15 symbols.

Next, the node representing combination $C = \{c_2, c_3\}$ is evaluated. Figure 4.6 shows the node's parse graph, for which a minimal parse results in the encoding $r_2, r_2, d, r_1, e, r_1, \$_1, b, c, \$_2, a, r_1, \$_3$ of

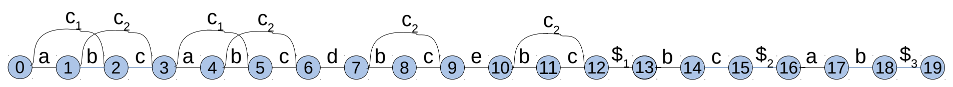

0 a 1 b 2 c 3 a 4 b 5 c 6 d 7 b 8 c 9 e 10 b 11 c 12 $1 13 b 14 c 15 $2 16 a 17 b 18 $3 19
c1
c2
c1
c2
c2
c2

Figure 4.5: An example parse graph for the input *abcabcdbcebc*, for the constituents *ab* and *bc*. The minimum encoding length is 15, and may be found by Minimal Grammar Parsing.

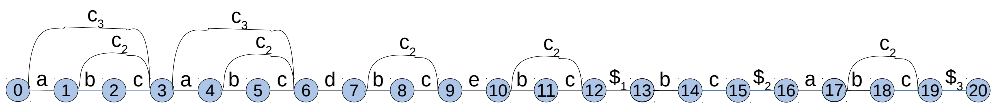

c3
c2
c3
c2
c2
c2
c2
0 a 1 b 2 c 3 a 4 b 5 c 6 d 7 b 8 c 9 e 10 b 11 c 12 $1 13 b 14 c 15 $2 16 a 17 b 18 c 19 $3 20

Figure 4.6: An example parse graph for the input *abcabcdbcebc*, for the constituents *bc* and *abc*. The minimum encoding length is 13, and may be found by Minimal Grammar Parsing.

length 13. Now both encoding lengths reachable in the *top-down* phase from the node representing $C = \{c_2\}$ are known – [15, 13] – it is again possible to select the first occurrence of the minimum, and check whether it offers an improvement on the smallest known encoding. At 13, it does not, and so the phase of the traversal switches to *bottom-up*, where each constituent in the current combination is removed to see if a reduction in $|G|$ occurs. Since the best combination known is $C = \{c_2\}$, for which only one move is possible to the node $\emptyset$ with encoding length 13, no further improvement can occur. At this point, ZZ terminates, and returns the constituent set $C = \{c_3\}$ as the observed optimum.

By performing a Minimum Grammar Parse using this constituent set, the following, final encoding can be obtained:

$a, r_1, a, r_1, d, r_1, e, r_1, \$_1, b, c, \$_2$

Note that, in this toy example, no optimisation of substring occurrences was necessary to discover the smallest grammar. Nonetheless, this simple example demonstrates the manner in which ZZ traverses the lattice of all possible constituent combinations, and also highlights that each candidate node in this lattice must be evaluated in either phase before a choice of destination node can be made.

4.8 Summary

This chapter presented a brief overview of straight-line grammars as approximations of Kolmogorov Complexity, along with a definition of their components. The compressors which will be employed within this thesis were outlined, and a description of the operation of the grammar-based compression methods IRR, IRC-OO and ZZ was given, highlighting the improvement which IRR-OO and ZZ offer over IRR schemes. A simple example of grammar construction was given for IRR-MC and ZZ, and the lattice representing the search space of all possible constituents which may be considered when constructing a grammar was described.

The following chapter will explore the application of ZZ to various analytical tasks given musical data, to demonstrate the relevance of grammars to such tasks, and evaluate the level of performance which may be expected. Later chapters will examine the role of parse graphs and Minimal Grammar Parsing in the production of more flexible grammars, and also the significance of evaluating all neighbouring nodes in the Lattice of constituent combinations during a ZZ traversal.

The true worth of an experimenter consists in his pursuing not only what he seeks in his experiment, but also what he did not seek.

Claude Bernard

5 Grammars in Application to Music Data: Initial Experiments

This chapter introduces a set of experiments conducted on symbolic musical scores, within which compression is employed as a process by which pertinent information may be obtained and used to perform a specific analytical task. The accuracy with which each task is performed is recorded, and the results are analysed in order to evaluate the degree to which compression, and specifically grammar-based compressors, are applicable and appropriate within the musical context. The purpose of the following experiments is to apply the existing works and methods discussed in Chapter 2 to a large, real-world data set, and explore the depth to which the theory that grammars can aid in musical analysis (Sidorov et al., 2014) is valid.

5.1 Introduction

Despite the existence of the works discussed in Chapter 2, such as the use of CFGs in the classification of pieces by genre, composer and style (Mondol & Brown, 2021), grammar-based compression algorithms have yet to be applied to a large, real-world collection of symbolic music, to discover how effective they may be on the wider set of musicological problems. In order to understand and evaluate the usefulness of grammars in relation to music analysis, it is necessary to define a group of tasks representative of common problems to which they may be applied, and to gauge performance against existing algorithms, including those designed specifically to process musical data. A large collection of digital scores is also required to obtain a true measure of response to widely varying input size, structure, and information content.

This chapter describes the wide-scale study of the application of grammars to three distinct types of analysis, using a collection of nearly 8000 pieces of symbolic music, and represents a significant contribution to current knowledge as the first large study of its kind using grammar-based compression. Its purpose is to investigate the hypothesis that grammars are capable of identifying musically significant structure in a score, which would allow their use in error detection and correction systems.

In order to evaluate the response of a grammar-based model to structural changes within musical sequences, both single and increasing numbers of alterations are made to elements of a score's pitch sequences. Each change is equivalent to the displacement of a note by a single staff line, such as might be found where an error has occurred during transcription, and the change this produces in the compressed model is measured. In an entirely novel approach, this response is used in the selection of candidate error positions, and its effectiveness is quantified and ranked over compressors used in previous musical studies, including grammar-based techniques. This chapter also introduces a novel method of discovering and correcting note errors which may be present in a musical score, and shows that grammar-based compression, without any additional domain knowledge such as the conventions governing the selection of key or scale, is able to isolate and suggest corrections for altered notes with an accuracy significantly greater than random chance.

The hypothesis that identification of musical structure is significant to the classification of folk tunes into families, where a family represents a set of closely related variants (since form within such families is likely to be similar), is also tested, along with pattern discovery and segmentation, as such patterns are by their nature significant and structural. To test this, it is necessary to gather a set of ground truth expert identifications from which accuracy may be quantified.

This study assesses the performance of grammars, given different musical representations to discover the range of possible response, against a well studied folk song classification task, the families of the 'Meertens Tune Collections' (van Kranenburg et al., 2016). The algorithms' ability to select patterns in a similar manner to a human expert is also tested, by evaluating performance on an official music information retrieval task from MIREX 2016, and calculating intersection over union for closest-matching grammar rules against segmentation by musicologist Siglind Bruhn (1993), for selected pieces from Bach's *Das Wohltemperierte Clavier*.

Although grammatical compressors do not possess musical context, the results show that they are able to perform musicological analysis, including more accurately identifying positive transcription errors than any other compressor tested. These are significant findings, and support the consideration of grammar-based techniques as viable alternatives to existing methods of music manipulation and analysis, such as those discussed in Chapter 2. This study represents the first extensive evaluation of these compressors on a large collection of music data, and extends the findings of Sidorov et al. (2014) to highlight previously unexplored possibilities when grammars are used as musical models. Alongside quantifying the ability of grammars to perform such orthogonal musical tasks, these novel experiments provide a foundation upon which the contributions of later chapters are constructed, and are also intended to show the potential for development of further grammar-based applications.

5.2 Method

The following experiments assess the performance of compression in three specific areas of music analysis. Following the hypothesis that a smallest model of a given digital score contains significant information regarding its composition, experiment classes which may leverage model features to solve orthogonal tasks are selected. Introduction of note errors which cause degradation of deliberate musical structure should also cause degradation in potential compression, resulting in larger, less compressed models. Musical similarity, such as between pieces of a common class, should be reflected by structure or content similarity within the model, allowing a metric such as compression distance to be used to cluster and therefore classify. Lastly, structures within the model should directly correspond with some structures defined during development of the music, since such grouping arrangements are common to both human composers and compressive explanations of input data.

Given these applications, it is now possible to identify common experimental elements. Each experiment class requires iteration over input scores, with some alteration of the data in the case of error introduction, or pairwise concatenation for distance calculation. At each step, compression of the data occurs, and resulting models and model sizes are recorded, and used to calculate further metrics where appropriate. Upon completion, results from each iteration are combined to produce an evaluation of success for each compressor. It is important to ensure input scores are composed of valid data, and consistent in representation. Since construction of a grammar from a large input may be highly complex in computational terms, it is also necessary to restrict experimentation to pieces for which the compression required for the experiment is feasible. Such a prediction may be made based upon the time required to produce a compressed model – or, indeed, the time needed to evaluate the addition of one constituent to a grammar-based model – as a measure of approximate complexity, as discussed in Section 3.6.

The study operates within this general framework, with specific parameters and features defined for each experiment, as detailed in Sections 5.4 – 5.6. A substantial collection of digital music scores is formed, with each piece processed into a common, easily processed format. The collection is catalogued, and compression times recorded so that pieces may be ordered by approximate complexity. For each experiment, pieces are selected from this catalogue in an iterative manner. Wherever possible, processing occurs in parallel to reduce the time required. Nonetheless, the computational complexity associated with compression and a high number of iterations prevent processing of the entire corpus during all tests; limitations are noted for each individual experiment.

5.3 Experiments: Error Correction, Classification and Segmentation

The following sections assess the capability of grammar-based compression to detect or correct errors in musical scores (such as those which may occur during transcription), classify folk songs by "tune family", and segment sequences a musicologically meaningful manner.

In Section 5.4, the response of compressors to the presence of single and multiple errors is investigated. Where response alone is evaluated in Sections 5.4.3 and 5.4.3, models are constructed from digital score data with and without changes which simulate basic transcription errors, and the length of their encoding is compared. A novel method of selecting candidate error positions is also presented in Section 5.4.3, where score data which contains an error is systematically altered, and models are constructed from the changed data, to examine whether a change which corrects an error is discernible from the compressor's response.

Section 5.5 applies grammar-based compression to the task of classifying the Meertens Tune Collections *Annotated Corpus v2.0.1* (van Kranenburg et al., 2016) by "tune family", where grammars are constructed from pairwise concatenations of pieces in the collection, and their encoding lengths used to compute a matrix of compression distances. 1-Nearest-Neighbour classification (Cover & Hart, 1967) is then performed, and the accuracy with which the process is able to classify the tunes evaluated with leave-one-out cross-validation (Kohavi, 1995). The purpose of this section is to assess the usefulness of grammar-based compression in the recognition of melodic similarity.

Finally, Section 5.6 explores the leverage of hierarchical structure present in models constructed by a grammar-based compressor to achieve musicological tasks where the aim is to delineate a score into musically-significant segments. In Section 5.6.3, the production rules present in grammars constructed from the music of the Johannes Kepler University Patterns Test Database (Johannes Kepler University, 2013) are considered to be "significant" to each piece, and passed into the publicly-available evaluation code for the MIREX 2016 *Discovery of Repeated Themes & Sections* task, whereafter the results are compared to those of the officially submitted algorithms. Section 5.6.3 compares the production rules of grammars built from eight pieces from Bach's *Well-Tempered Clavier* Book I to those identified by the musicologist Siglind Bruhn (1993), to assess whether such grammars are capable of segmentation in a similar manner to a human expert. The last experiment in this chapter, described in Section 5.6.3, explores the possibility that manually altering the production rules of a grammar constructed from a musical piece may provide useful assistance when editing a score to change or improve it, or to create an entirely new composition with the same hierarchical structure.

5.4 Applying Grammar-based Compression to the Prediction of Data Errors

5.4.1 Purpose

These experiments are designed to test the hypothesis that a grammar-based compressor may be used to detect the presence of data errors within a musical score. Any such error is likely to degrade the regularities present within the piece, reducing the ability of the compressor to exploit regularity and causing production of a larger grammar. Similarly, correction of an existing error should, in many cases, allow production of a smaller grammar. A musical "spell-checking" system might be based on this technique, automatically locating incorrect data arising from OMR or human transcription errors. The experiments are constructed to demonstrate a relationship between data errors and compression strength, measured here as model size.

Ideally, compressor response to a range of real-world errors would be evaluated, with instances of each error type drawn from observations based upon the study of Optical Music Recognition, and the work of scribes. However, the approach adopted here does not allow for a wide range of possibilities to be tested, and, as such, these experiments are designed only to explore compressor response to a single type of error, so that a great many inputs may be tested and an overall measure of ability taken within the confines of the task. To achieve this objective, a data error resulting in the displacement of a note by ±1 interval has been selected. As noted by Bainbridge & Bell (2001) in their discussion of the difficulties associated with OMR, many different types of error can occur which result in the alteration, commission or omission of note symbols. Insertion or deletion of a symbol in a compressor's input is very likely to result in the generation of a compressed model of a different encoding length, but when a symbol is altered a change in encoding length will only occur if the symbol was useful during compression. Stutter (2020) described human transcription errors within D-W Cod. Guelf. 628 Helmst. (*"Magnus Liber Organi; W1"*) (Baxter, 1931), where notes appear displaced a musical third from their intended position, detectable due to the unusual dissonance created. An exploration of OMR systems (McPherson, 2006) such as CANTOR (Bainbridge, 1996) showed it was not uncommon for an automatic transcription system to erroneously offset a note by a single step on the staff where its notehead is in contact with another immediately below or above it, thus altering the bounding box used to identify its position sufficiently to associate its centre with a neighbouring line or space. Burlet et al. (2012) proposed an online music notation editor which made use of human input to correct errors introduced during transcription, including the correction of notes which the OMR system had been placed 1-2 vertical positions from their written noteheads. As such, displacement of notes by a single staff position is reasonable to represent an instance of a possible transcription error, in order to investigate the possibility that structure present within a grammar built from a symbolic musical sequence has a relationship to correct pitch structure within that sequence.

5.4.2 Method

The performance of ZZ and IRR-MC are here compared to three general-purpose compressors, as a basis for what may be expected from standard tools. LZW, Burrows-Wheeler Transform with run-length encoding, and GZIP are selected for this study. BWT and LZ-derived algorithms have been shown to perform well on tasks involving the approximation of Kolmogorov Complexity (Kolmogorov, 1963), and produce a clearly defined symbolic output, the size of which may be easily computed and compared to an encoded grammar. Model size for ZZ and IRR-MC is defined in Equation 4.1. For LZW, model size is taken as the sum of the length of the alphabet and encoded output, plus a separator symbol between them. For BWT with RLE, model size is taken as the sum of the length of the encoding of each symbol and the number of times it repeats. For GZIP, model size is taken as the number of bytes necessary to store the compressed output on a filesystem. Because these measures are not directly comparable, the ratio of compression achieved is computed instead.

These experiments are computationally expensive. Evaluation of the change in ZZ model sizes to an increasing number of errors (Section 5.4.3) in each representation for Bach's Fugue No. 10 from *Das Wohltemperierte Clavier* Book I, shows that sequences of note intervals in the diatonic scale produce the strongest general response (Figure 5.1). Based on this result, performance on a diatonic interval representation of music alone is evaluated, as described in Section 3.3. A "ground-truth" model is produced by compressing this diatonic data. A model containing n errors is produced by selecting n non-boundary symbols within the data, and altering them before generating a compressed model; selected values are changed by ± 1 interval, to simulate a common single staff-line transcription error. The subsequent interval value is also changed so that the following notes remain unaffected.

Positions where alterations will occur are chosen at random from within each input, in an attempt to prevent the measurement of response to a specific or uniform set of circumstances, such as might be

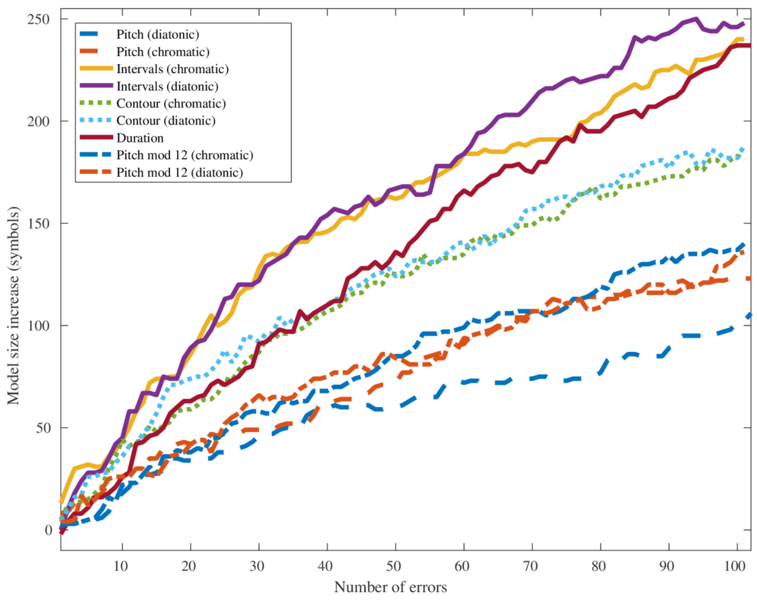


Figure 5.1: Reaction of grammar-based compressors to an increasing number of errors, in all available symbolic representations, for Bach's Fugue No. 10 from *Das Wohltemperierte Clavier* Book I. On average, diatonic intervals produce the strongest response.

observed if a degree of correlation existed between the chosen positions and the structure of a score (for example, if the n_{th} element of the sequence were consistently selected, but had a high probability of falling within a strongly-repeating motif where it would be more likely to cause a reduction in potential compression). The use of entirely random values aids in the measurement of a general response, such as that which would be observed over a large population of scores.

Compressor response is measured by computing the difference in model size for ground-truth and altered data. Ability to correct errors is measured using precision, recall and F-measure, with the following definition:

- *True positive:* a correction of a previously altered value.
- *False positive:* alteration of a previously correct value.
- *True negative:* no alteration of a previously correct value.
- *False negative:* a failure to correct a previously altered value.

Table 5.1: F-measures: Accuracy of model size response to error.

Experiment	ZZ	IRR-MC	LZW	BWT	GZIP
1 (3107 pieces, asc. comp. time)	0.79	0.77	<u>0.81</u>	n/a	0.72
2 (all pieces, 25% of each piece tested)	<u>0.81</u>	0.79	0.73	0.60	0.72

5.4.3 Experiments

Response to a single Error

For each piece, the unaltered data is first compressed, and the size of the model measured. Then, for each position selected as described in the following paragraph, a single error is introduced into the data, generating a compressed model, and its size is measured. The difference in model size is taken as the response of the compressor to this single error.

Two variations of the experiment are performed. In the first, all values within the piece are altered, but for a limited number of pieces since experiment run-time in this case is *tl* for each piece, where *t* is grammar build time and *l* is input length in symbols (as shown in Figure 3.2), making a test of the entire corpus impractical. In the second, only 25% of each piece is considered for candidate error positions, which allows every piece in the corpus to be tested.

Results

The results are presented in Table 5.1.

The results show that all the compressors tested respond to the presence of a data error, regardless of the difference in compressor strength or output encoding. Hypothesis testing, with the null hypothesis that the result values obtained for each pair of methods may come from the same distribution, confirms that the results for each method belong to Student's *t*-distribution (Student, 1908) and are statistically distinct from each other given a significance level of 5%. Result distribution is presented in Figure 5.2, where ZZ and IRR are seen to possess similar characteristics.

The simple run-length encoded Burrows-Wheeler Transform gives the weakest response, followed by GZIP. The grammar-based methods show good performance, with ZZ consistently outperforming IRR, as expected.

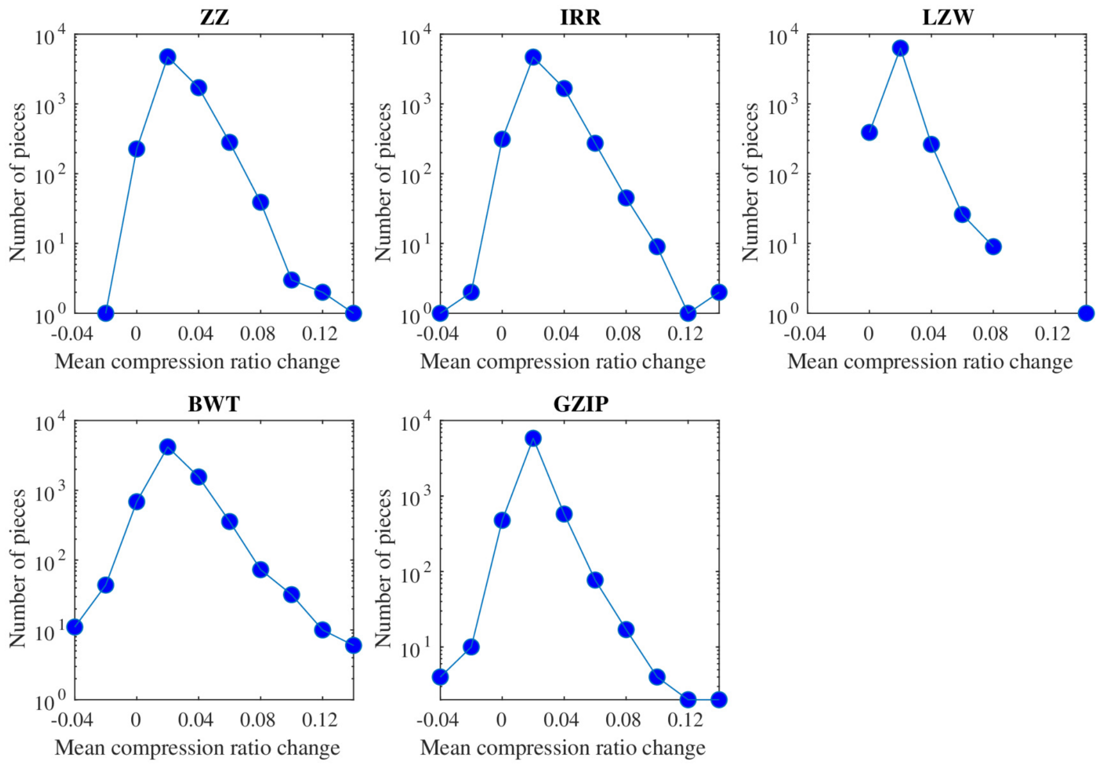


Figure 5.2: Histograms highlighting the overall trend in response for each compressor. Each figure shows the distribution of mean compression ratio change observed for each piece, following the introduction of a single data error. Note that the LZW figure contains three ratio groups which contained no pieces ($y = 0$).

Despite LZW being a poorer compressor in comparison to statistical coding techniques (Shanmugasundaram & Lourdusamy, 2011), it exhibits the most sensitivity to errors in the data when applied to short musical sequences. This occurs because LZW is able to take advantage of short, rarely repeating sequences within an input, whereas a grammar requires a greater overall gain before such sequences may be chosen as constituents for the model. Due to this fact, LZW is responsive to single errors within a greater proportion of each piece. Since constituents chosen for inclusion in a grammar might be considered structurally significant, the primary interest is in the compressor's response to errors within these segments.

ZZ response over tested positions for two scores may be seen in Figures 5.3 and 5.4. The rule hierarchy chosen by the compressor is plotted above each note sequence in *S*. The average change in grammar size for an interval at a given position in the input sequence, resulting from the independent introduction of a ± 1 interval error to each voice, is plotted at the top of each figure, with standard deviation shown as grey shading. Averaging model size changes for all voices in each figure was done to enable the display of the overall response of the model per interval; where no deviation exists, an identical response occurred across all voices.

It is important to note that the simplicity of the experiment may itself introduce a degree of bias. Alteration of a single symbol, representing a diatonic interval, causes the loss of an instance of that symbol within the input sequence, and adds to the count of its substitution. Additionally, the following position in the sequence undergoes the same process, resulting in a worst-case difference of 4 in the count of unique symbols present within the input's distribution, producing a likely imbalance of the input in its altered state. However, since pitch is represented as diatonic intervals, the value chosen by each alteration is likely to be present within the majority of inputs, and where this is untrue the input will contain characteristically few unique symbols, and should indeed alter noticably when an error is introduced; only where the original interval represented an extreme of the same sign as the introduced error would the alteration be certain to introduce an entirely unique symbol. With regard to the grammar construction process, it is certainly not impossible that the alteration may reverse the effect of an intended state in such a way as it becomes *more* useful during compression; for instance, changing a final note in a pattern which deviated from its template due to an approaching harmonic change might result in its return to an exact match to the template. Alteration of a set proportion of each input sequence means that fewer symbols will be changed for smaller inputs, which in turn reduces the degree to which their associated response may be taken as representative of the population of inputs, although since a large sample of such inputs was tested this effect is softened. Overall, there is a minimal risk of bias strongly affecting the experiment's outcome, and the response seen remains indicative of the potential to detect a change in structure. Nonetheless, the results should be considered

Term	Instances (Non-overlapping)
0,0	3
0,0,0	2
1,1	1
1,0	3

Table 5.2: Repeating terms within the vector [0,0,0,-2,1,1,1,-4,1,2,0,0,-1,1,0,1,0,0,0,1,0].

with the limitations highlighted by these observations in mind.

LZW Error Sensitivity What is the reason for LZW's success in some circumstances? Two significant factors are its compound use of repeating terms, and the lack of a score function to exclude sub-optimal terms from its dictionary. This causes *any* reduction in repeating terms to affect the size of the model resulting from LZW's processing. ZZ and IRR, on the other hand, only add constituents which decrease overall model size at each iteration, thus it is possible for structural alterations to occur in small inputs without affecting the grammar hierarchy. For these reasons, LZW is the most responsive of the three compressors for this class of input.

As an example, consider the following vector ($|v| = 21$):

$$v = [0, 0, 0, -2, 1, 1, 1, -4, 1, 2, 0, 0, -1, 1, 0, 1, 0, 0, 0, 1, 0]$$

Repeating terms within v where $|t| > 1$ are shown in Table 5.2.

During dictionary construction and encoding, LZW uses all but the second term:

$$v = [0, \underline{0, 0}, -2, 1, \underline{1, 1}, -4, 1, 2, \underline{0, 0}, -1, 1, 0, \underline{1, 0}, \underline{0, 0}, \underline{1, 0}]$$

A single reference is required for each replacement, giving encoding $|e| = 15$.

ZZ, however, is unable to perform any such replacement. Each constituent used would require one rule reference per instance, plus its length including terminator symbol, in the encoding. This results in the gains shown in Table 5.3

ZZ is unable to compress this input, instead returning the original vector as an encoding, giving $|e| = 22$.

If an error is introduced in any position which prevents the use of a repeat instance, ZZ will still be unable to compress. However, LZW continues to use all remaining repeats, giving $|e| = 16$:

$$v = [0, \underline{0, 0}, -2, 1, \underline{1, 1}, -4, 1, 2, 3, 0, -1, 1, 0, \underline{1, 0}, \underline{0, 0}, \underline{1, 0}]$$

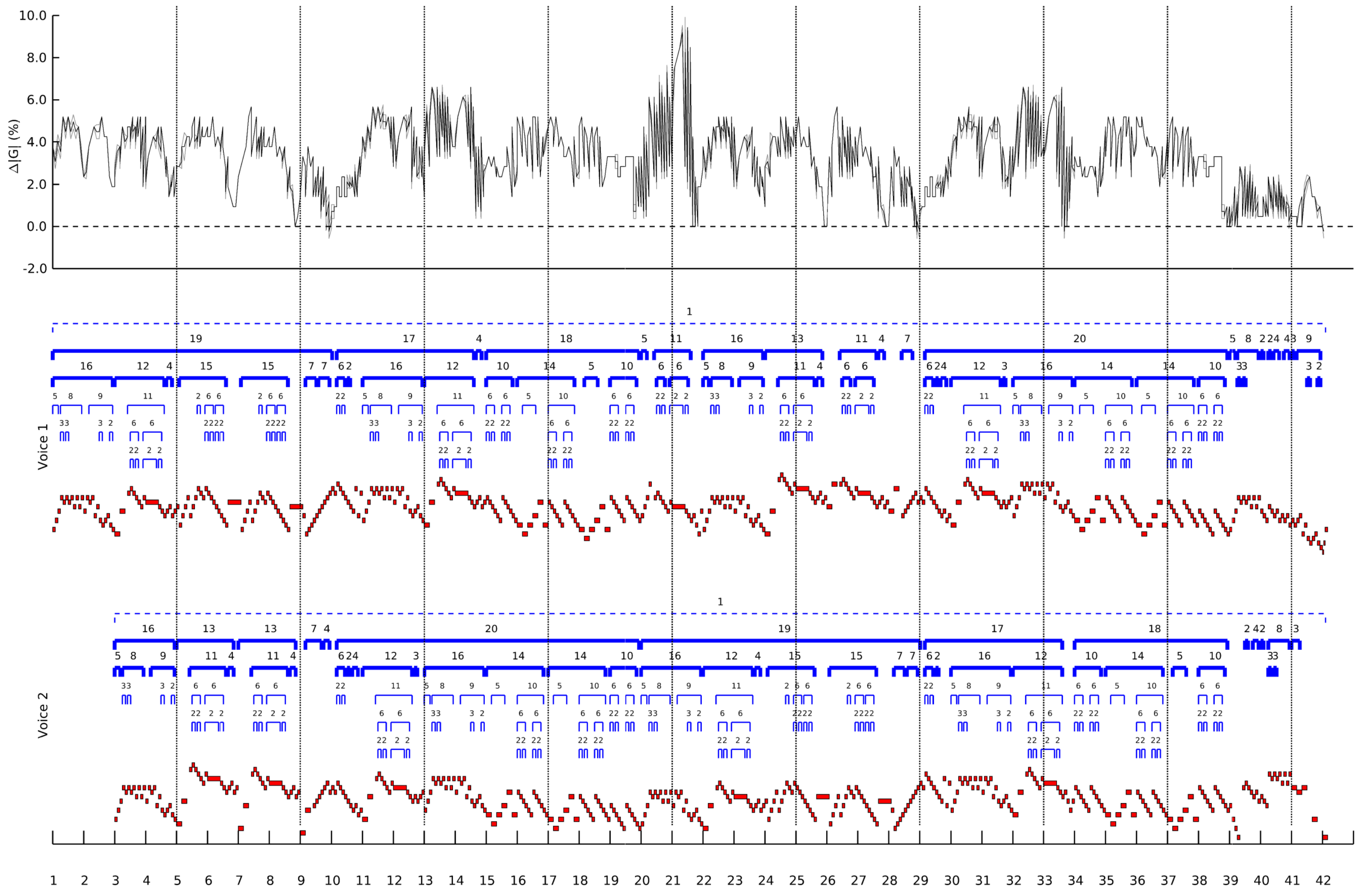


Figure 5.3: Response to data errors over two passes of all note positions using ZZ, for Bach's Fugue No. 10 from WTC I.

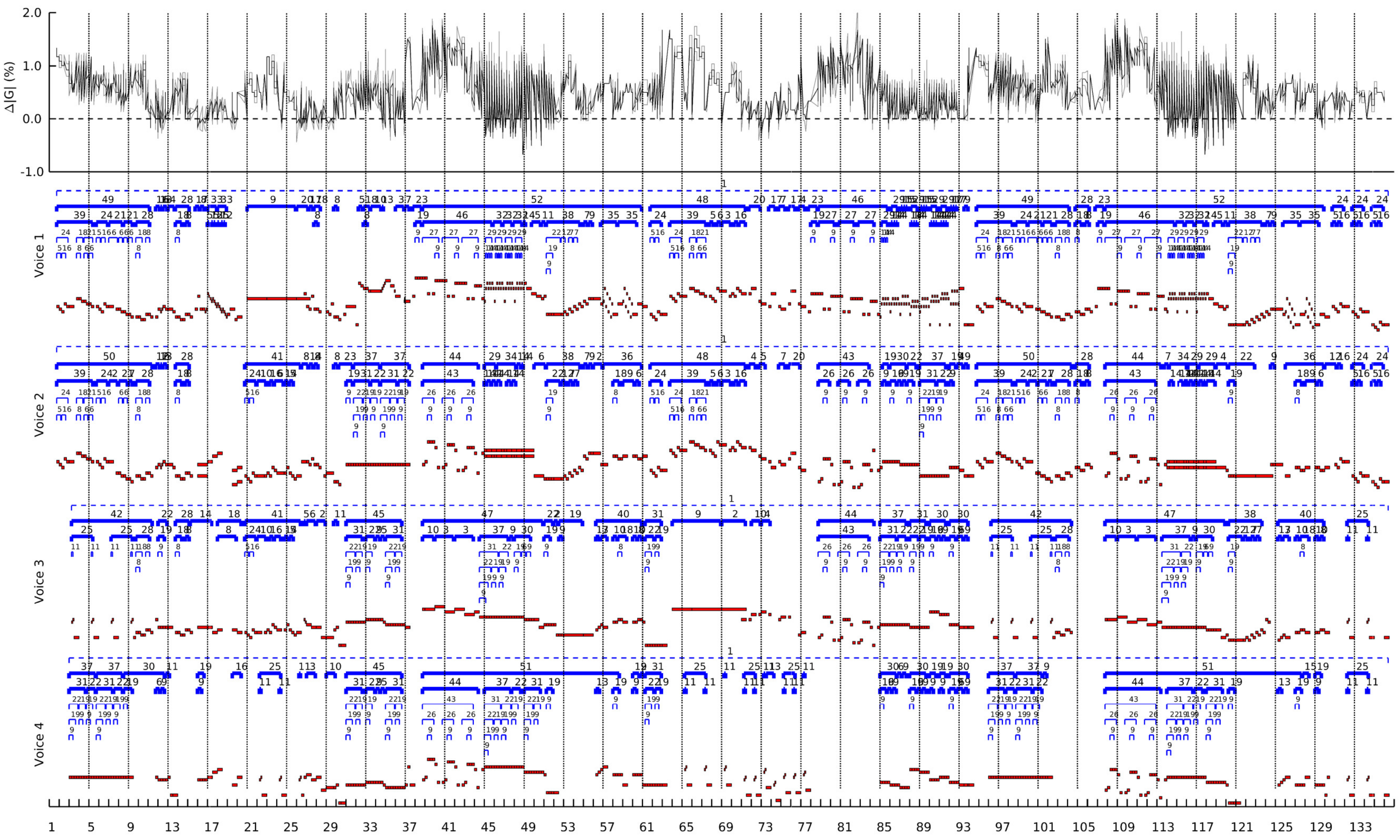


Figure 5.4: Response to data errors over two passes of all note positions using ZZ, for the Finalé of Haydn's String Quartet No. 22 in G major, Op. 17 No. 5. A dense and shallow hierarchy can be seen in the first voice in bars 85-93, suggesting structure of less significance to an exact compressor during this period. Musical content here is varied and unique within the score, producing an expected reduction in compression, in clear contrast to the majority of the piece.

Term	Gain	Reference cost	Inclusion cost	Total gain
0,0	6	3	3	0
0,0,0	6	2	4	0
1,1	2	1	3	-2
1,0	6	3	3	0

Table 5.3: Overall gain / cost, of replacing terms within the vector [0,0,0,-2,1,1,1,-4,1,2,0,0,-1,1,0,1,0,0,0,1,0] with referenced production rules.

Thus LZW is responsive to minor structural changes which ZZ ignores.

Response to an increasing number of Errors

For each piece of length l, a set of nl symbols to alter is selected from a uniform distribution, choosing $n = 0.5$ as early experiments showed no significant response could be clearly detected beyond this threshold. Errors are then introduced at $0 \leq p \leq nl$ positions, generating a compressed model at each iteration, and its size is measured. A change in model size from the model for $p = 0$ is taken as the response of the compressor.

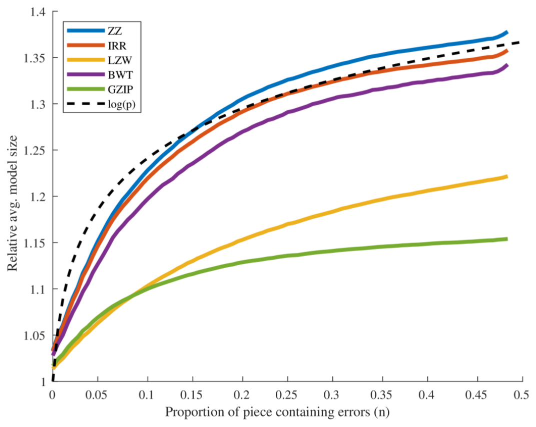
Pieces were split by length into three groups, $l = 1$–200, 201–1000 and ≥ 1001, so that the response to inputs of different lengths could be observed. Since distribution of piece length within the corpus is non-linear, a strong decline in the number of data points comprising the average occurs as e increases. For this reason, only results for smaller counts should be considered an accurate representation of response.

Results

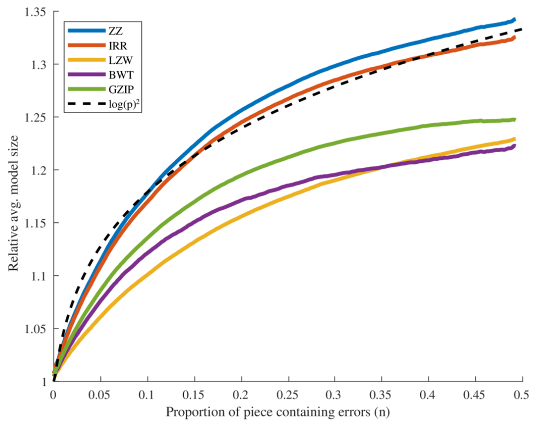
The results are presented in Figures 5.5a – 5.6c.

Different compressors produce differently encoded models, and so the sizes of their models may not be directly compared. To aid the attempt to compare compressor performance, the compression ratio each model achieves is calculated. Hypothesis testing confirms that results for each method belong to Student's t-distribution given a significance level of 5%, but not all results within each group are distinct from each other. For the group 1–200, the null hypothesis is true between ZZ, IRR and BWT. Where pieces are of length ≥ 1001, ZZ and IRR are not distinct from each other, however it is important to note that the group's sample size is markedly small at 134 pieces, which may account for the failure in differentiation.

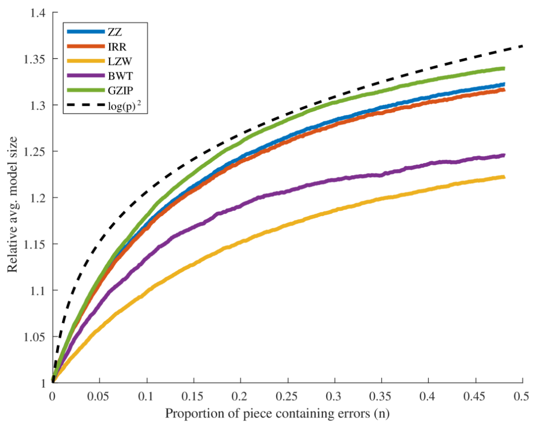
All piece groups exhibit a detectable rise in model size as the number of errors increases, until approximately $p \geq 0.25l$, where a significant proportion of the input has become corrupted; such re-



(a) *Group 1: pieces of length 1–200*



(b) *Group 2: pieces of length 201–1000*



(c) *Group 3: pieces of length 1001+*

Figure 5.5: Average response of each compressor to an increase in the proportion of each piece containing errors, over three length-based groups.

sponse is supportive of the hypothesis. Of the three groups (lengths 1–200, 201–1000, $\geq$ 1001, Figures 5.5a – 5.5c), the first two then show a reduction in response with the 1–200 group levelling, perhaps as structure within the input data becomes degraded to that of noise. The group formed from pieces of length $\geq$ 1001 does not exhibit a decreased response within the range tested, $p \leq 0.25l$. Instead, model size continues to grow, suggesting this larger input data contains more structure which may be compressed, and is therefore more error-sensitive.

All groups show a similar response, but with more instability as group sample size decreases, as might be expected. Each existing error simultaneously raises the probability of encountering noise, and also reduces the available structure from which to discriminate errors. This causes the reduction in response with increasing errors, resulting in a plateau where detection is no longer possible. Determination of the point at which this effect begins for pieces of high complexity is left for a future exercise.

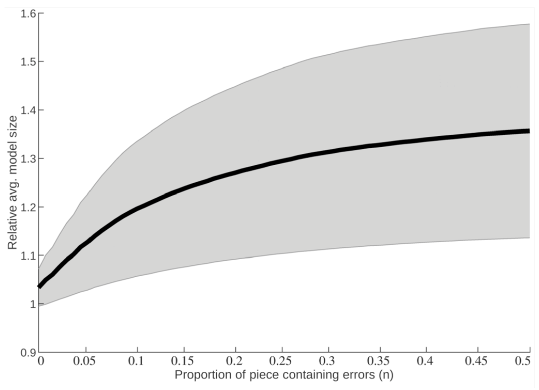
Perhaps the most significant result is the clearly superior performance of GZIP in the group containing pieces of length $\geq$ 1001, where it responds with greater sensitivity than all other methods; for pieces of length $\leq$ 1000, ZZ offers the best performance. This suggests ZZ and GZIP are best able to compress and therefore leverage structural information from smaller and larger pieces respectively.

Standard deviation is significant, with each method showing a highly similar response; for this reason, plots for ZZ alone are included here (Figures 5.6a – 5.6c). Greater variance as p increases is to be expected, since errors in this context may introduce different structure as well as degrade that which exists. However, it is notable that an overall decrease in compression occurs in all cases, demonstrating the response of the compressor to overall and not simply local structures. Variance has an apparent relationship to p, with the greatest effect visible in the group of largest pieces. The plot for the 201–1000 group contains some local maxima, correlating with common piece lengths within it. Overall, such large variance means no method in this study may be relied upon to show a strong response to given errors within music data, but since all methods exhibit decreasing compression a response will certainly occur, albeit minor in magnitude.

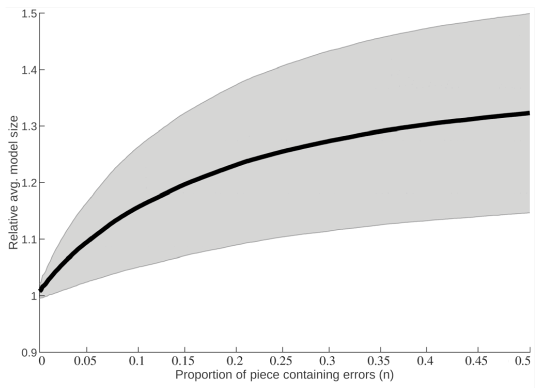
Automatic selection of candidiate Transcription Error Positions

Building on the previous experiments, a novel method capable of automatically selecting notes believed to be errors is now presented, designed as an aid to the process of transcription error correction.

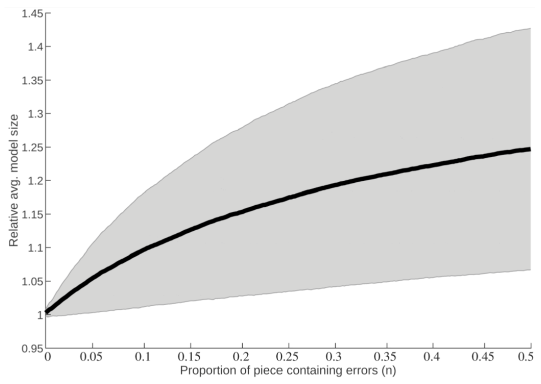
For each piece, $c = mp$ representations are created, where p represents the number of positions which will be altered, and m represents a proportion of each containing a single error in one of the c positions. A compressed model is then constructed for each representation, with its size taken as a



(a) *Group 1: pieces of length 1–200*



(b) *Group 2: pieces of length 201–1000*



(c) *Group 3: pieces of length 1001+*

Figure 5.6: Average response of ZZ to an increase in the proportion of each piece containing errors, with standard deviation, over three length-based groups.

Table 5.4: F-measures: Correct selection of candidate error position; pieces are tested in ascending order of computation time.

Experiment	ZZ	IRR-MC	LZW	BWT	GZIP
m=n=1 (565 pieces, 100% of each piece tested)	<u>0.22</u>	0.20	0.19	n/a	0.16
m=0.5, n=0.25 (565 pieces, 25% of each piece tested)	<u>0.27</u>	0.24	0.26	0.15	0.20
m=0.5, n=0.25 (5735 pieces, 25% of each piece tested)	<u>0.35</u>	0.32	0.21	0.12	0.24

baseline for a version of the piece containing an error.

For each such version, $p = nl$ positions are individually altered by ± 1 interval. Exactly one such change will correct the error in that version. A compressed model is then constructed for each potential correction, and the model's size measured. Following the hypothesis that a musical piece which contains errors has a degraded structure and is therefore less compressible, the size of the resulting model is compared with that of the version containing an error. Any alteration which results in a smaller model size is taken as a likely successful correction, and identification of a candidate transcription error position.

For the c versions of each piece into which an error was introduced, precision, recall and F1 are calculated for each alteration, to evaluate the method's performance. Any size smaller than the baseline is taken as a positive, and any greater than or equal to the baseline as a negative. Two versions of this experiment are conducted, the first with $m = n = 1$, the second with $m = 0.5$ and $n = 0.25$, as preliminary experiments showed this would allow a reasonable proportion of the corpus to be processed within the available time. The second experiment is repeated on the same pieces used for the first, to provide an indication of the difference in observed performance when testing a smaller proportion of each piece.

Results

The results are presented in Table 5.4.

Hypothesis testing again confirms that results for each method are statistically distinct from each other. Result distribution is presented in Figure 5.7, where ZZ and IRR can be seen performing consistently well.

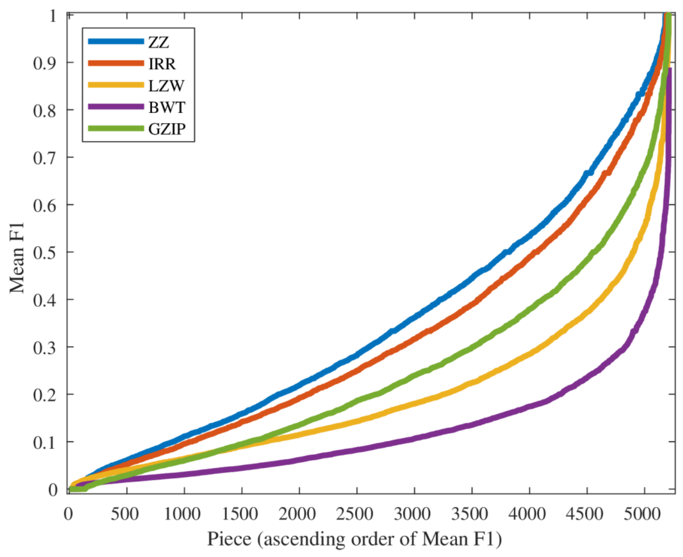


Figure 5.7: Distribution of mean F-measure per piece for each compressor (in ascending order).

These figures again suggest a relationship between the strength of each compressor and its performance on this task. When provided with more complex data for the second experiment, the ability of GZIP to compress with greater strength than LZW (Savakis, 2000) is clear, and BWT shows poor accuracy, likely a result of reduced compression. ZZ consistently produces the best result ahead of IRR, which is by comparison a naïve method. Perhaps most interesting is the clear advantage both grammar-based methods exhibit.

An overall increase in F-measure can be seen to take place with $m = 0.5$, $n = 0.25$ instead of $m = n = 1$, partly due to the increased probability of correctly selecting a candidate selection from the smaller option set. This limitation is also present in the test results from the larger group of 4390 pieces; actual performance will be poorer than shown if pieces are processed in their entirety.

Although direct comparison of compressors by the length of their encoded representations is not possible without full consideration of encoding differences, it is interesting to note that average piece compression ratios, as may be seen in Table 5.5, generally support the hypothesis that greater compression results in best performance for this application.

Method Practicality

Although theoretically interesting, selection of candidate error positions in this manner is a computationally complex task, and potentially impractical. If t corrections per position are to be tested, given ZZ complexity $g = O(n^5 \times m^2)$, where $n =$ number of symbols in the input sequence and

Table 5.5: Average ratio of compressed to uncompressed data for pieces in both experiment groups, for each compressor.

Experiment	ZZ	IRR-MC	LZW	BWT	GZIP
m=n=1 (565 pieces)	0.86	0.88	0.86	n/a	1.13
m=0.5, n=0.25 (5735 pieces)	0.89	0.89	0.87	0.838	0.728

m = number of constituents per node in the lattice (Meredith, 2014), and substring search complexity $s = O(n^2)$, the computational complexity of this method is $O(t(n^7 \times m^2))$. However, this upper bound is rarely reached in practice, and the experiment highlights the superior ability of grammar-based compressors over the tested algorithms to identify musically incorrect structure. Despite the relatively low F1 scores achieved overall, the results are stronger than could be expected by chance, showing that the compressor exhibits a definite reaction to the degradation of structure within the musical sequences. This supports the hypothesis that note errors can indeed be detected by a grammar-based compressor, since a measurable accuracy is observed, as would be expected if the hypothesis were true to any degree.

5.5 Classification

In this experiment, ZZ and IRR-MC are applied to the task of classifying the Meertens Tune Collections *Annotated Corpus v2.0.1* (van Kranenburg et al., 2016) by "tune family", as defined by expert musicologists from the Meertens Institute. Selecting this widely attempted task provides an opportunity to examine the performance of the grammar-based method in the context of many published studies. Its success is evaluated for eight individual musical representations, and when these are weighted and combined to classify each piece in the collection, overall performance is calculated.

5.5.1 Purpose

This experiment is designed to test the hypothesis that the computed compression distance between grammar-based models for two musical scores may represent an approximation of their similarity, and may therefore be useful in the classification of scores by pairwise distance. An ideal grammar construction algorithm will select the set of patterns which, when replaced within the input, produce the smallest model. Thus, where a pattern exists in both scores, it provides more potential for compression than a pattern unique to one score, and compressing scores with common components is likely to produce smaller models than those generated from dissimilar pieces.

5.5.2 Method

For a given compressor C, scores are selected in a pairwise fashion. For each piece, eight strings are produced, one for each individual representation described in Section 3.3 except *note in diatonic octave*, resulting in $x_1, x_2, \ldots, x_8$ for the first piece, and $y_1, y_2, \ldots, y_8$ for the second piece. For each pair of representations, three models are constructed: $C(x)$, $C(y)$, and $C(xy)$, where xy represents a concatenation of x and y separated by a unique symbol. A Normalised Compression Distance (Li et al., 2004) is then computed for each pair of scores, as defined in Equation 5.1:

$$\mathrm{NCD}(x,y) = \frac{C(xy) - \min(\ C(x),\ C(y)\ )}{\max(\ C(x),\ C(y)\ )} \tag{5.1}$$

For each representation, 1-Nearest-Neighbour classification (Cover & Hart, 1967) is used to cluster scores by distance, with leave-one-out cross-validation (Kohavi, 1995) used to evaluate "tune family" prediction accuracy against expert-defined ground truth. The known overall success rates for each representation are then used as weights in combining class predictions for each score, and their sum used to arrive at the final class. Success rate r is calculated as follows, where c is the number of correct predictions, and t is the total number of pieces tested:

$$r = c/t \tag{5.2}$$

5.5.3 Results

The results for each individual representation using ZZ are shown in Table 5.6. Success rates for each compressor when combining weighted representations is shown in Table 5.7.

Of the individual music data representations used, chromatic-based pitch interval vectors produces the greatest success rate, with both pitch and pitch-interval vectors generating the strongest response from the types tested. This is perhaps unsurprising; a chromatic representation retains all available pitch detail, and use of intervals offers some degree of invariance to repeating patterns which are transposed within a piece. Since the MTC *Annotated Corpus v2.0.1* contains short strophes (average length is 48 notes), often in a single key, it is reasonable to expect transpositions to be less important than in longer works. This may be reflected in the good performance of both pitch vectors. The duration vectors provide least success during classification; distribution of note lengths within

Table 5.6: Per-representation success rates from NCD classification of the MTC-ANN v2.0.1 using ZZ.

Representation	Success rate
Intervals (chromatic)	0.88
Pitch (diatonic)	0.87
Pitch (chromatic)	0.87
Intervals (diatonic)	0.85
Octave note (chromatic)	0.76
Contour (diatonic)	0.69
Contour (chromatic)	0.68
Duration	0.63

Table 5.7: Rate of successful classification of pieces from the MTC-ANN v2.0.1 for ZZ and IRR-MC.

ZZ	IRR-MC
0.92	0.83

Table 5.8: Classification success rates achieved by various methods on the MTC-ANN v2.0.1.

Work	Avg. success rate
Kranenburg et al. (van Kranenburg et al., 2013)	0.99
Stober (Stober, 2011)	0.98
Conklin (Conklin, 2013a)	0.97
Goienetxea et al. (Goienetxea, Neubarth, & Conklin, 2016)	0.96
Louboutin & Meredith (Louboutin & Meredith, 2016)	0.94
Hillewaere et al. (Hillewaere, Manderick, & Conklin, 2014)	0.94
Volk & Haas (Boot, Volk, & de Haas, 2016)	0.93
This work	*0.92*
Velarde et al. (Velarde, Weyde, & Meredith, 2013)	0.84
Meredith (Meredith, 2014)	0.84

the *Annotated Corpus v2.0.1* is highly skewed, with one duration alone accounting for over half of all instances, and another for 30%. This causes a reduction in information from which to characterise each tune family, a likely reason for this representation's loss of accuracy.

Despite performing well, ZZ is not able to improve upon the techniques used in some existing studies, most notably van Kranenburg, Volk, and Wiering (2013) who achieved an accuracy of 0.99 using a combination of Inner Metric Analysis, pitch, and note phrase-offset features. Table 5.8 provides a comparison of classification results on the *MTC-ANN v2.0.1* for selected studies.

Within the context of these studies, the grammar-based method performs as may be expected of a sequence-based similarity model without domain knowledge. Kranenburg et al. computed rate of success when interval sequences only were used, achieving an average of 0.92. There, transposition was used to place each pair of scores into a common key, and the Needleman-Wunsch alignment (Needleman & Wunsch, 1970) balanced to minimise the penalty for continued shifting of a pattern segment when alignment is sought. This allowed for flexible pattern matching, akin to human-like recognition of simple musical variations. It is possible that this strategy provided a performance gain similar to that obtained from this study's use of multiple representations, suggesting that the addition of flexible matching may further improve the method. However, verifying whether this is the case is left for later chapters.

Stober (2011) adopted a generalised approach, creating a distance measure based on the weighting of various domain-based facets, such as chords and harmonies. In contrast to the average success rate of 0.99 from van Kranenburg at al. (2013), he achieved an average of 0.97 where the class of the query piece is unknown. The grammar-based method's lower success rate is expected given the lack of domain knowledge employed, and perhaps it is reasonable to suggest that use of pitch-based facets alone for Stober's method might cause a drop in accuracy similar to that seen with these constraints in the study by van Kranenburg et al. (2013).

Studies by Conklin (2013a) and Goienetxea et al. (2016) reported average success rates of 0.97 and 0.96 respectively, both higher than the method presented here, and also incorporated domain knowledge in the form of viewpoints, each representing features derived as a function of note pitch, duration and onset time. In particular, Goienetxea et al. (2016) employed a reductive heuristic based on the novelty of common patterns, and both works suggested that features such as motifs, metric, and phrase information are important in addition to pitch for this task. The increase in accuracy seen in this experiment when distance is based on a weighted combination of representations supports these assertions, and the greater success of all these studies where higher level musical features are used suggests that such an addition might also improve the accuracy of a grammar-based method.

Work by Louboutin & Meredith (2016) employed morphetic pitch alone (Meredith, 2006b), and reported an average success rate of 0.85 where COSIATEC was used to process a single viewpoint, similar in informational terms to where this study operates on diatonic pitch vectors. Their method outperformed the grammar-based approach where different compressors, including LZ77 (Ziv & Lempel, 1977), were given multiple viewpoints and the results combined. Their work highlights the suitability of LZ77 in the analysis of polyphonic music, and the potential gain from leveraging compressed models of various types, a strategy which might also improve this study's method. Hillewaere et al. (2014) employed a flexible approach, using Levenshtein distance for pairwise alignment of both melodic and rhythmic data, but without compressed modelling. They achieve an average success rate of 0.94, compared to 0.92 by van Kranenburg et al. (2013) using the same representation, suggesting a superiority of edit distance over Needleman-Wunsch, and highlighting a link between flexible patterns and musical variations.

Interestingly, Boot et al. (2016) reported best results for sequence alignment of uncompressed data, where 0.93 is achieved using a note-to-note correspondence comparable to that employed by van Kranenburg et al. (2013). It is important to note that all cited studies related to the latter work demonstrate high success rates despite making use of fewer musical representations than this experiment, showing that higher level modelling of features may be more significant to tune family classification than a wider combination of lower-level data. However, the grammar-based approach improves on results from Volk & Haas when using compressed representations, perhaps pointing to the superiority of grammars in isolating patterns significant to classification of the MTC.

Velarde et al. (2013) recognised that their Haar Wavelet analysis method underperformed in comparison to string-matching approaches when classifying the *Annotated Corpus v2.0.1*, and showed that the use of chromatic pitch instead of a scale-based representation such as morphetic pitch limited effectiveness. However, chromatic pitch provides the best success rate in this experiment. It is possible that combining results from wavelet analysis of multiple representations could also present an improvement.

In conclusion, existing studies suggest the performance achieved on this task is to be expected when using grammar-based compression of such sequences as the basis of a distance metric. The addition of flexible matching which better captures musical structure, higher level feature modelling, and, perhaps, combining grammar-based distances with those of other compressed models is likely to offer a measurable improvement.

5.6 Segmentation

5.6.1 Purpose

This experiment is designed to test the hypothesis that, following the Minimum Description Length principle (Rissanen, 1978), generating a model of music data using a grammar-based compressor may cause division of that data to occur in a musicologically significant fashion, resulting in structure which is similar to that which a human expert would define for the piece. The ability to automatically segment a score into meaningful segments could have several applications. Primarily, an accurate analysis of a piece may be obtained very quickly, providing an aid to academics and performers and a potential insight into the intentions of its composer. Indeed, the differences between an automatic and human-made segmentation may themselves highlight overlooked interpretations, or simply an alternative perspective, consideration of which may benefit musicological knowledge. Such a model might also be employed as a compositional aid, allowing alterations in flow and structure to be made to a score at a high level, but using meaningful "units" of note sequences. These experiments are designed to demonstrate the degree of similarity between computationally-derived structures and human analyses, and the potential for high-level score manipulation, when grammar-based compression is employed.

5.6.2 Method

The diatonic interval representation is selected for these segmentation experiments, encoding each piece as a concatenated sequence of voices as described in Section 3.3. For the examination of grammar-assisted editing, chromatic intervals, note onset intervals and durations are chosen, since all notes

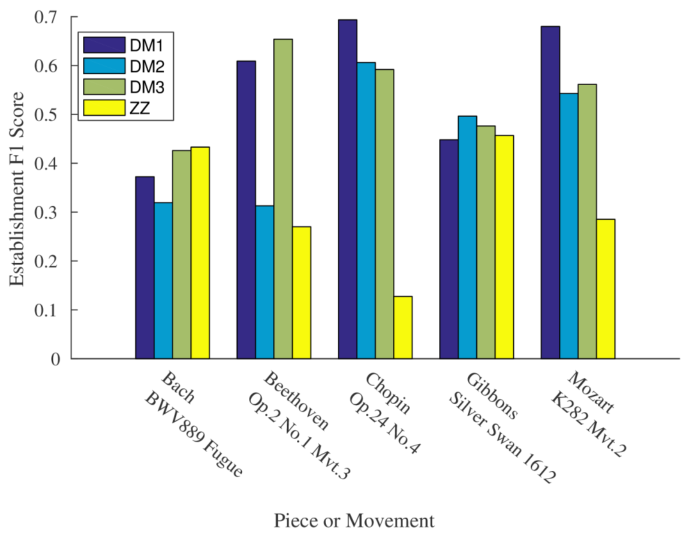


Figure 5.8: MIREX 2016 Discovery of Repeated Themes & Sections symPoly task, Establishment F1 score for 2016 algorithms DM1, DM2 & DM3 with additional ZZ results.

within a score may be modelled using this combination. In each experiment, ZZ is then used to construct a grammar for each piece, and all sub-rules within S are selected and processed using the following experimental methods.

5.6.3 Experiments

MIREX 2016 Discovery of Repeated Themes & Sections task

Grammars are built from each of the pieces in the Johannes Kepler University Patterns Test Database (Johannes Kepler University, 2013). Each sub-rule of S is passed to the MIREX 2016 code designed to evaluate algorithm performance on the symbolic, polyphonic *Discovery of Repeated Themes & Sections* task. Focus is given to two of the available metrics: *establishment* and *occurrence*. *Establishment* is a measure of an algorithm's ability to identify any instance of a ground truth pattern, whereas *occurrence* measures its ability to identify all instances within a piece. As defined by the evaluation procedure, matches with a score threshold ≥ 0.75 are selected as positive identifications. F-measures are calculated from each metric, and these are compared to the official results for 2016.

Results

The results are shown in Figures 5.8 and 5.9.

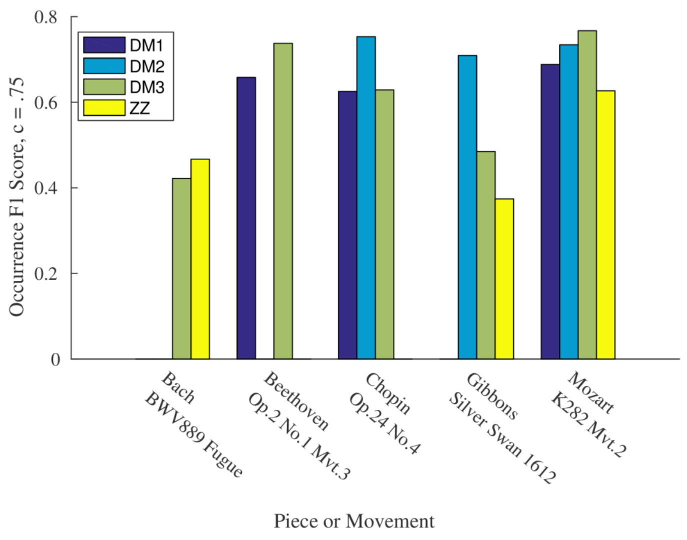


Figure 5.9: MIREX 2016 Discovery of Repeated Themes & Sections symPoly task, Occurrence F1 score for 2016 algorithms DM1, DM2 & DM3 with additional ZZ results. A score of 0 results from failure of an algorithm to identify at least 75% of the total instances of any pattern - no bar is plotted for these cases.

ZZ outperforms all other methods in identifying patterns for Bach's Fugue No. 20 from *Das Wohltemperierte Clavier* Book II, and improves on the poorest method when seeking any instance of ground truth patterns within Gibbons' *The Silver Swan*, although it fails in retrieving all instances. On all other pieces in the dataset, it responds most poorly, in particular failing to identify pattern sets within the performance threshold for Beethoven's Op. 2 No. 1 Movement 3, or Chopin's Op. 24 No. 4.

ZZ's strong performance on the Bach Fugue may be attributed to the frequent repetition of its subject, for which exact sequence matching is most suitable. Algorithms DM1–3 are based on SIATEC (Meredith et al., 2002), an algorithm capable of flexible matching when applied to inexact sequences of significant similarity. This makes them more suitable for use with musical data containing variations, such as the pieces by Beethoven and Chopin. The suggestion that grammars built on exactly repeating patterns are unsuitable for such data may be reasonably supported by ZZ's strength in pattern establishment over identification of instances; difference of a single symbol within a pattern instance causes ZZ to split the entire sequence around this symbol, whereas SIATEC simply omits the point from its pattern definition, resulting in the latter's greater ability to retrieve all pattern instances.

Structural Analysis of Bach's *Well-Tempered Clavier*

Given the strong response obtained to the Bach Fugue on the MIREX 2016 task, the response of grammar-based compression to Bach works is investigated further, in an attempt to evaluate whether stuctures present within the grammar may directly relate to those deemed significant by a human expert in a specific musical analysis. As discussed in Chapter 2, a variety of approaches exist to the analysis of music, an automated approach such as grammar-based segmentation is clearly conducted very differently to a holistic understanding built from years of human study and experience. Different approaches may also result in widely different structures, and the simple comparison made in this experiment is intended only to examine the musical significance of the structure returned by a grammar-based compressor in the context of a single interpretation. Nonetheless, an algorithm which can automatically provide a segmentation which aids in the analysis of music is clearly a useful tool.

The first eight pieces from Bach's *Das Wohltemperierte Clavier* Book I are selected for use in this experiment. For each piece, a set of segments is first defined as *(start, end)* offsets into the sequence of intervals used to represent it. Each segment is defined by the following process:

- Where Bruhn(1993) shows a definite start and end point for a given repeating section, the intervals representing these sequences are located within the input data.
- For each instance of these sequences, *(start, end)* pairs are defined for each *exactly* repeating subpattern, since the chosen compressor operates only on groups of exactly repeating symbols.
- Each set of *(start, end)* pairs is labelled following Bruhn's description, and considered a "ground truth" unit which it is desirable for an automatic segmentation tool to identify.

Grammars are then built for each piece, and a set of *(start, end)* pairs created for each of its subrules in S, by iteratively expanding the grammar and recording the offsets of each rule occurrence within the input sequence. Where rules may be obviously grouped, such as two consecutive non-terminal symbols occurring beneath a single span specified by Bruhn, an artificial rule containing these group elements is manually added to the grammar.

A score is then greedily computed to represent match degree between the ground truth segments and the grammar's rules. For each ground truth segment, the sum of Jaccard Distance to instances of each grammar rule is calculated, and the rule at minimum distance is considered a unique match to the target segment. Where no match exists, distance is set to maximum. The overall match between a set of expert-derived segments and the grammar rules which most closely represent them is taken as the mean Jaccard Index for all such segments.

Table 5.9: Mean Jaccard Index to Bruhn's analyses of J.S. Bach's *Das Wohltemperierte Clavier* Book I, No. 1–4.

WTC I		
No.	Prelude	Fugue
1	0.82	0.86
2	0.87	0.95
3	0.91	0.78
4	0.66	0.62

Results

The results are presented in Table 5.9.

Figures 5.10 and 5.11 show the hierarchy produced by ZZ for WTC I Fugue No. 2 and Prelude No. 3, in "piano-roll" format.

These figures show matching typical of that returned by the experiment. Rules are selected from various levels of the hierarchy, yet generally where a human-chosen span occurs there exists a grammar rule of markedly similar length and offset, and a strong correlation between them is predominant. Each ground truth section within Fugue No. 2 almost exactly aligns with a grammar rule in the first three layers of the hierarchy, with 64% of matches occurring at the top level. Some low level rules begin with an additional symbol over the target sections, causing the reduction in accuracy. Although such symbols form part of exactly repeating sequences, they exist beyond Bruhn's boundaries, and thus outside what this experiment considers accurate against the ground truth. There are some weak results, in particular for No. 4 Prelude and Fugue.

The grammar for Fugue No. 4 exhibits good correlation, but with important exceptions. For example, several instances of the first subject are not matched to any candidate rule. Two distinct circumstances explain these omissions. In some cases, intervals within the subject exposition differ, either because of a change in musical scale or by deliberate variation, such as a transitive note linking a subject to the following phrase. This shortcoming might be addressed by incorporating domain knowledge into the grammar construction process, so that sequences containing variations may also be considered as repeating. In other instances, the compressor has not chosen a rule matching a subject span; instead, a rule producing greater compression has been formed using a symbol also present within the subject, thus preventing use of an existing rule which defines the subject alone. It is conceivable

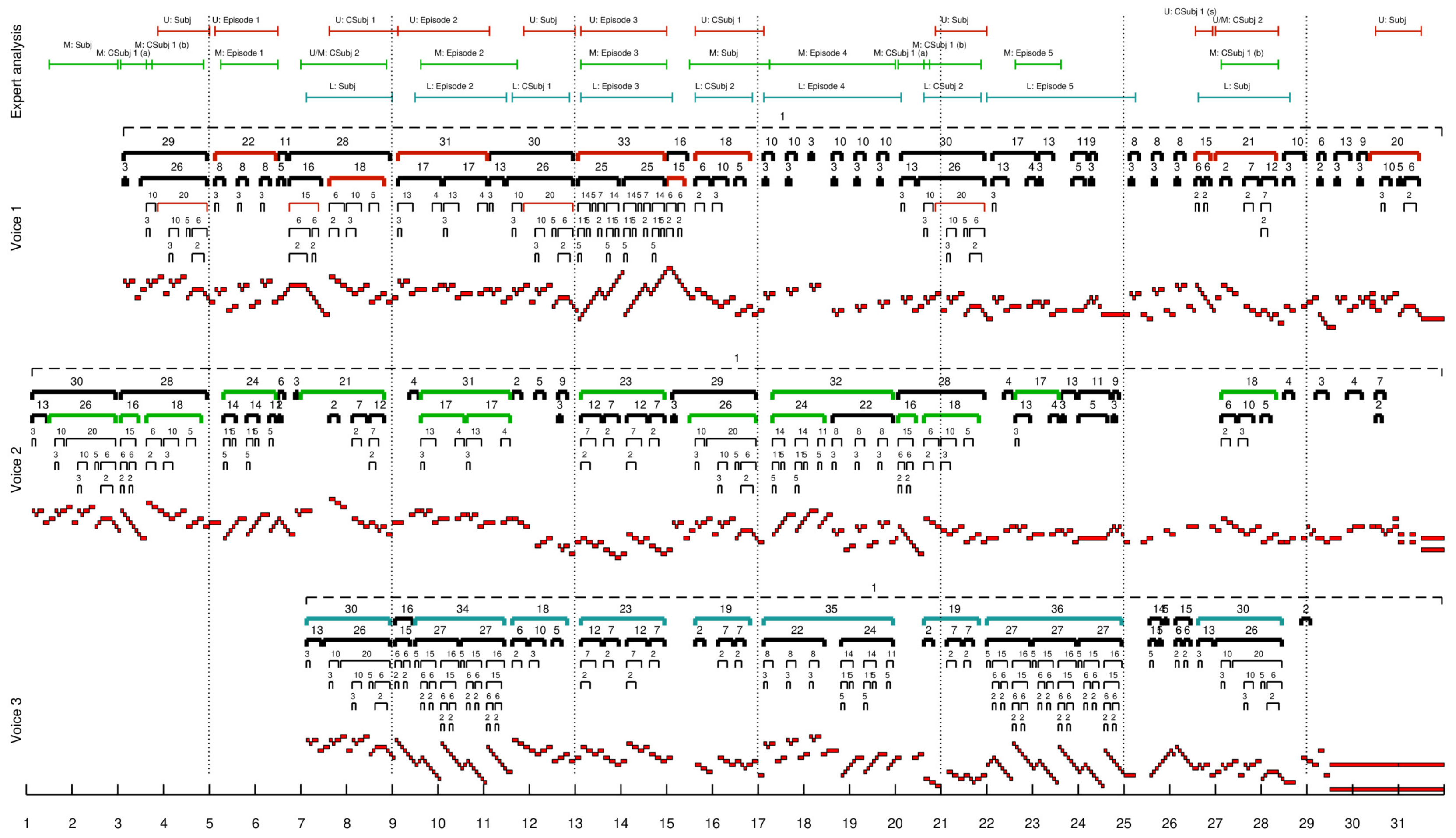


Figure 5.10: Comparison between musicologist-identified segments and rules within the hierarchy returned by ZZ, for Bach's Fugue No. 2 from WTC I. Jaccard Index for this piece is 0.95. Note the erroneous selection of rule 15 in voice 1, bar 15; simply selecting the rules most strongly matching the target sections can result in inclusion of non-matching segments, where the penalty for this choice does not outweigh rejection of the rule. However, both rule 15 instances do indeed match partial counter-subject motifs, suggesting that algorithmic identification of a repeat not highlighted by Bruhn has in fact occurred.

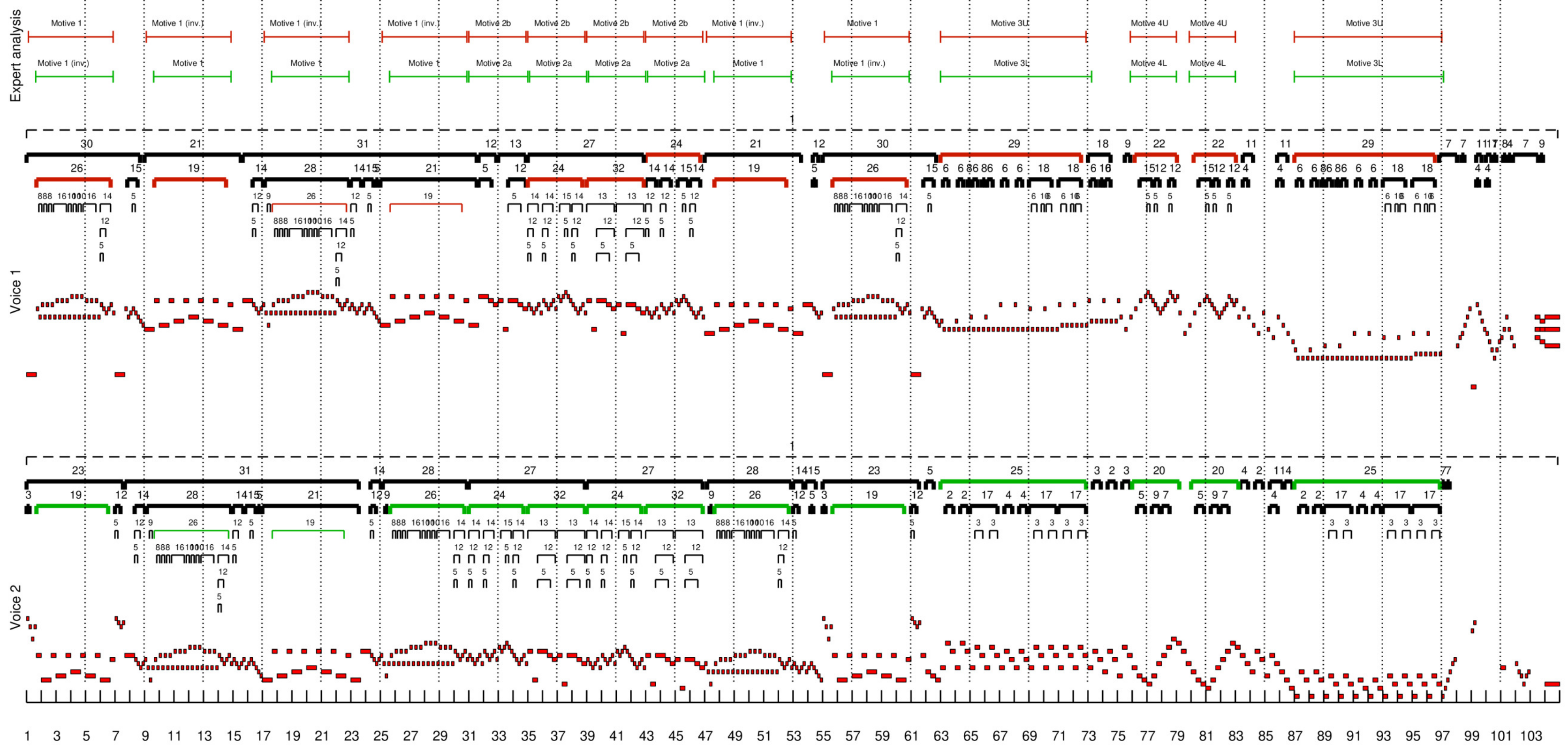


Figure 5.11: Comparison between musicologist-identified segments and rules within the hierarchy returned by ZZ, for Bach's Prelude No. 3 from WTC I. Jaccard Index for this piece is 0.91. Voice 3 contained only closing notes, and is omitted from this figure.

this might occur because a smallest model was not produced by ZZ. However, it may be more reasonable to suggest that overlapping structural explanations exist for this passage, such as may be seen in Bruhn's analysis of Book I Prelude No. 1 (1993). This is a more serious shortcoming: the branching of a grammar's hierarchy prevents the modelling of intersecting structures, which may represent important complementary explanations in a musical analysis.

The greedy approach to choosing grammar rules does not restrict matches to a single level of the hierarchy, and the most closely correlating rules are not always those from the top level. For example, in Figure 5.10, rules 16 and 18 are selected together as the strongest candidates for counter-subject 1, part b (bars 3–5 and 20–22). However, rule 28 exists at a higher level and contains both chosen rules, suggesting that extension of Bruhn's segment by a single interval might be a good representation of this counter-subject instance. Mismatches such as these may indicate a failure of the model to recover a musicologically ideal segmentation, or present an alternative explanation of a score's structure. The rule hierarchy may indeed provide an advantageous view of a given segment at various levels of abstraction, and hint at the manner in which a composer employs compound motifs and techniques when creating the piece.

It is important to note the small sample presented here. Digitisation of expert analyses is time consuming and open to interpretation, and access to a wide collection of digital interpretations from various analytical schools against which to evaluate alogrithmic methods is currently unavailable – such a resource could significantly benefit similar research. Accepting this limitation, the results suggest the level of correlation between grammar rules and expert-defined segments is notable, and likely aided by the highly-structured nature of the analysed pieces, making identification of exactly-matching repeats relevant and useful.

Grammar-assisted Editing

To demonstrate some benefits and disadvantages of a grammar-based editing system, a simple editing tool was developed, choosing Bach's Prelude No. 1 from *Das Wohltemperierte Clavier* Book I as input since it has a clearly-defined structure against which alterations can be easily distinguished. A single grammar is built for this piece, and its non-*S* rules are altered with the intention of producing a musically reasonable output. In this case, an attempt is made to simply reverse the ascending motif used throughout. Finally, the grammar is expanded to produce edited score data, which may be represented visually or played.

Edit operations are restricted to the substitution of individual terminal symbols within rules representing pitch intervals, to demonstrate the effect of changes to individual pitch values only, and avoid

Table 5.10: Changes applied to rules of the grammar modelling the chromatic intervals of J.S. Bach's Prelude No. 1 from *Das Wohltemperierte Clavier* Book I.

Rule	Instances	Original intervals	Edited intervals
2	9	3, -3	-9, 9
3	22	3, 5	12, -4
4	7	4, -4	-8, 8
6	4	-7, 4, 3	-3, -4, 7
7	3	-6, 3, 3	-3, -3, 6
9	7	0, 0, 0	0, -7, 7
10	4	3, -10, 7	-7, -2, 9
11	3	3, -8, 5	-5, -4, 9
13	4	5, 7, -12	12, -7, -5
15	2	-6, 2, 4, -6, 2	12, -4, -2, -6, -4
19	10	5, 4, -9, 5, 4	9, -4, -5, -3, 12
20	4	5, 5, -10, 5, 5	10, -5, -5, 10, 0
21	8	7, 5, -12, 7, 5	12, -5, -7, -5, 17
22	8	-10, 4, 6, -10, 4, 6	-6, -4, 10, -6, -4, 10

the alterations in structure which are likely to occur if non-terminal symbols are changed. Edits were further constrained so that the sum of each rule remains unchanged, to avoid introduction of a pitch offset for subsequent notes.

Results

The rule hierarchy produced by ZZ from the unaltered score is shown in Figure 5.12, represented as a two-voice piano-roll.

Terminal symbols in 14 of the 29 rules of the pitch grammar are changed; Table 5.10 provides details of the edits made.

The frequencies with which rules 2, 3 and 19 occur in the expanded score are high, indicating these are important foundational elements. Indeed, altering them has a strong effect across the entire piece, highlighting that many related changes may be made simultaneously using this method. As a human editor, care must be taken to preserve the context of each rule. For example, a rule containing only scale degrees 1 and 5 may appear in a major and minor setting, and introduction of a 3rd can cause dissonance where the rule coincides with the opposite scale. Auditioning after each rule is edited will

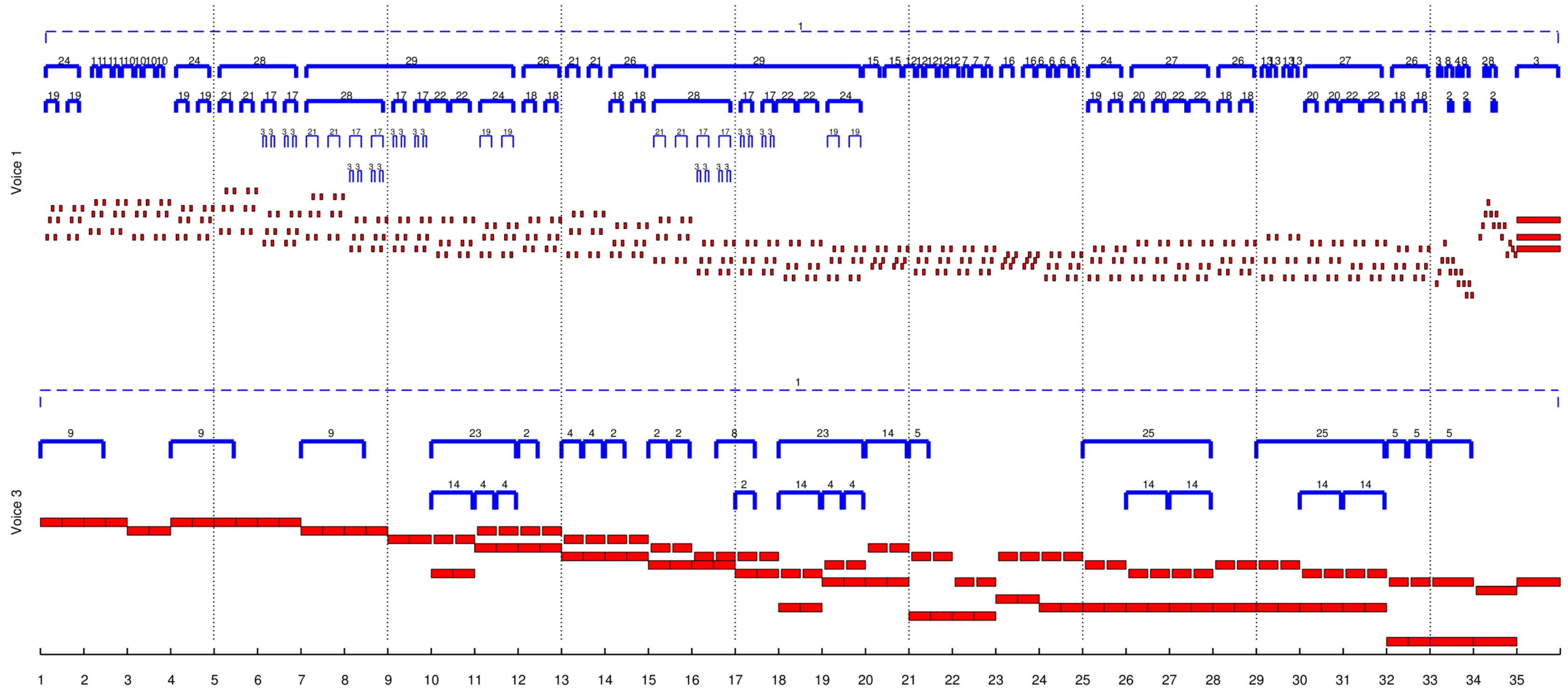


Figure 5.12: Structure of grammar for Bach's Prelude No. 1 from WTC I, built from chromatic intervals by ZZ.

allow a manual check to be made, and undesirable changes to be reversed.

These simple edits allow fast production of a believable score, where the goal of reversing ascending figures is mostly achieved. However, for two-symbol rules such as 2 and 3 this is not possible, since only one pitch change may be made before returning to the original note; in these cases, inversion of the motion was chosen. Enforcing the constraint that the rule's sum must not alter for rule 20 also requires a compromise: its final interval cannot be a negative value, which breaks the descending pattern it otherwise maintains. Rule 9 represents four identical bass notes, and a 5th was substituted for the third note in the sequence to show additional movement is easily possible whenever the change does not violate the context of the rule's occurrence. A segment of the resulting score is shown in Figure 5.13.

Several disadvantages to this approach can be seen. Since pitch and rhythm are represented separately as sequences in S, insertion or deletion of a single element results in misalignment between note attributes, and will occur as many times as the rule is used during grammar expansion. Encoding voices in separate sequences as described in Section 3.3 does not support the modelling of harmonic relationships such as chords, and where related notes are edited these relationships must be manually preserved. The ability to make multiple changes from editing a single symbol may also result in dissonant combinations which are not immediately obvious, as a single rule can exist in several different musical contexts. Care must be taken not to allow the effect of altering a rule to impact subsequent rules and values, especially where interval-based data representations are chosen.

However, grammar assisted editing naturally enables changes specifically relating to content or structure to be made, either individually as in this experiment, or in combination. Where the musical context of a rule is known, for example the scale its pitch values belong to, it may be altered to contain anything within that context. Large-scale modifications to a score are possible via rule editing, from alteration of low-level building blocks containing only terminal symbols to strong structural changes through manipulation of rules containing a deep hierarchy. Binding information such as pitch, onset and duration together symbolically could help address some of the disadvantages affecting such operations. Since, as shown in the previous experiments, a grammar's rules may represent a musically significant segmentation of a piece, it is reasonable to suggest this grammar-based method allows edits to occur within a contextual, musical framework.

5.7 Summary

In this chapter, the application of grammar-based compressors to six practical musical applications was investigated, and their performance compared to that achieved by the use of other popular compres-

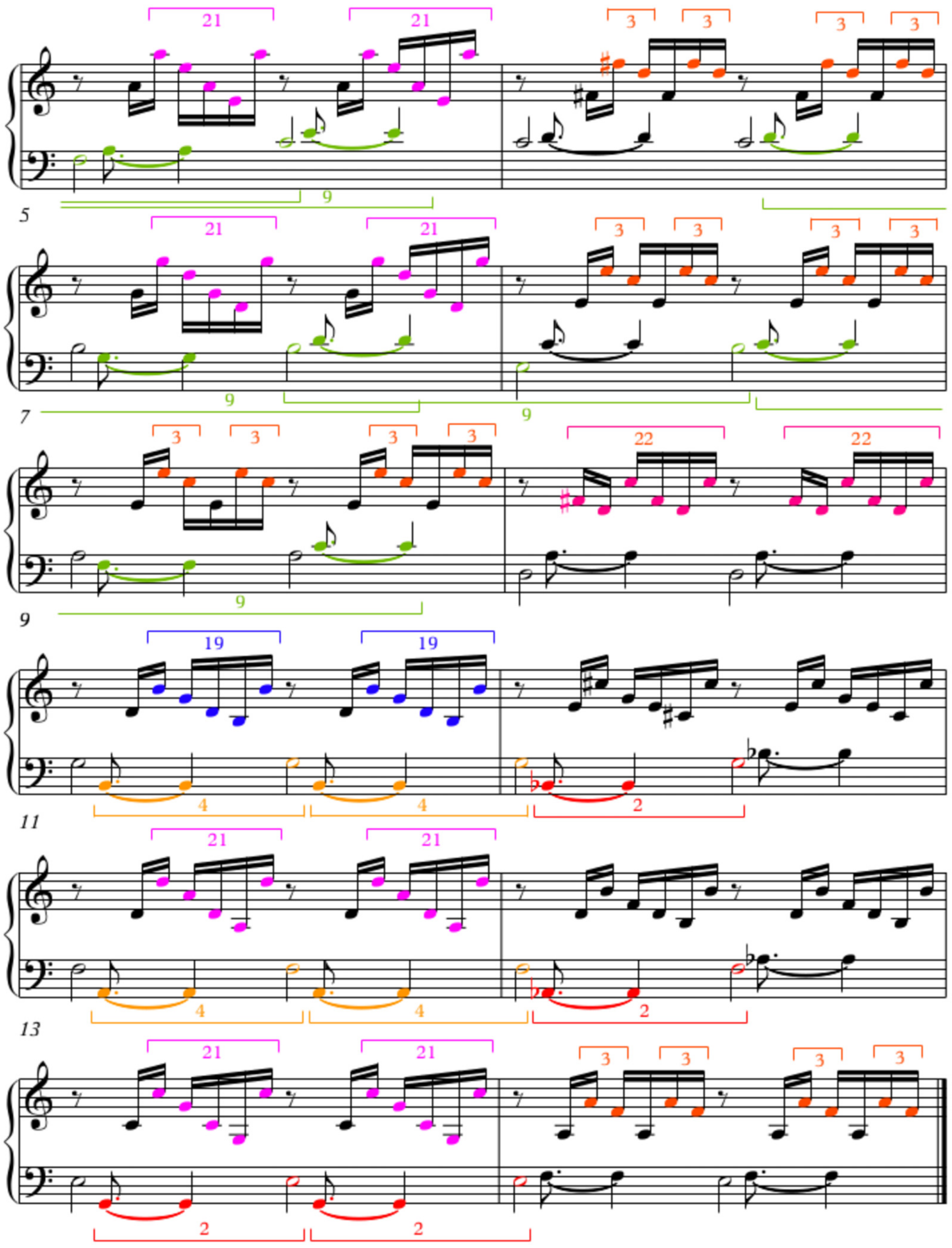


Figure 5.13: Bars 5-16 of Bach's Prelude No. 1 from WTC I after grammar-assisted editing.

sion algorithms. The responsiveness of each method to errors was examined, and their performance measured when applied to the location of errors, classification of folk music by tune family, and the discovery of expert-defined musicological patterns.

When tasked with detection of a common transcription error, LZW proved most responsive for pieces too small to compress by standard grammar, but beyond this margin ZZ proved most sensitive. All methods showed a logarithmic response to an increasing number of errors, with GZIP outperforming ZZ as input length became significant. ZZ was most successful in correctly identifying the position of a single error, with an F-measure of $0.22 - 0.35$. Strong variation in response was measured for all methods, showing none can be relied upon to respond correctly in each individual case. However, every method generated a larger model in the majority of cases.

When grouping pieces from the Meertens Tune Collections by tune family, ZZ was able to perform moderately by comparison with existing studies, when the results from multiple representations were weighted and used in nearest-neighbour classification. ZZ was also applied to the discovery of expert-defined patterns from the polyphonic *Discovery of Repeated Themes & Sections* task presented in MIREX 2016, where it bettered all submitted methods for a highly-structured Bach fugue, but gave unstable results for the other four scores, highlighting the greater flexibility of SIATEC-based algorithms in discovering inexactly repeating patterns. Finally, exactly repeating structures identified by musicologist Siglind Bruhn within eight works from Bach's *Das Wohltemperierte Clavier* Book I were compared to rules within grammars produced by ZZ for each piece. Although the results showed wide variation, a strong correlation was observed at high levels of the hierarchy, a notable achievement considering that ZZ possesses no domain knowledge.

The results generally support the link between strength of compression and the information recovered, as suggested by the Minimum Description Length principle (Rissanen, 1978). They also show that ZZ can outperform several popular compressors when applied to detect degradation in musical structure and classification of Dutch folk tunes, in the latter case when provided with attribute-rich note data. However, exact grammars cannot rival current techniques when seeking expert-defined patterns containing variations, and can fail to generate desirable rules where an overlapping explanation, and therefore rule, exists. These findings highlight the significance of intersection in the analysis of musical compositions, and support suggestions by existing studies that the ability to abstract musical features and pattern templates using domain knowledge is important to algorithmic analysis; such additions are likely to improve the performance of ZZ on these applications.

In the following chapter, a method is presented by which the transformation of patterns may be incorporated into a grammar-based model during its contruction, thus allowing fundamental equivalence within the input to be handled despite the presence of incidental variations, in order to enable the inclusion of domain-specific features.

We are considering on how to build on the efficient algorithms developed in the field to capture more such rules.

Payam Siyari & Matthias Gallé, 2017

6 Grammars allowing Rule Modification

THIS chapter presents a novel addition to straight-line grammars, which allows them to contain production rules that may be modified during expansion. This enables an increase in substring similarity during their construction, which, by creating an opportunity for rules to perform greater replacements within the input string, allows the production of smaller grammars. This is achieved by the introduction of "transforms" which are encoded within the grammar itself, and may apply to any of its rules when their associated reference symbol follows a non-terminal symbol. As with non-terminals, the expanding program is designed to recognise the transform's instructions and perform the desired operation.

A novel aspect of the scheme is the ability for the transform to be customised, so that both generalised and domain-specific operations can be encoded, and may be included or excluded by the grammar construction program depending on whether they prove useful in generating a smaller encoding – thus, it is possible for them to be leveraged following the Minimum Description Length principle (Rissanen, 1978). This introduces a "dimension" of rule modifications, in addition to the existing dimension of substrings, which much be traversed during grammar construction. The following sections introduce and describe the key concepts behind the contribution, and suggest a method of selecting constituents and transforms based upon the existing optimisation method ZZ (Carrascosa et al., 2011).

6.1 Introduction

A classical straight-line grammar for a given input string makes use of a dictionary of terms which exactly repeat within that string. A number of terms are chosen as candidates, and where these occur within the string they are replaced with references to production rules. The result may be an encoding of fewer symbols in length than the original string, due to the replacement of many instances of a given group of symbols with a single reference symbol per occurrence, with only a single instance of that group as overhead within the encoding.

Each term, represented within the grammar as a production rule, may be said to represent the "template" from which replacements are made during expansion of the grammar back into the original string. During that process, it is possible to augment the replacement in some fashion, i.e. to modify the string the production rule generates, such that every instance of the rule's output occurring within the expanded string is *not* identical. An additional symbol, assigned specifically to indicate the modification, may be included immediately following (or preceeding) any rule reference. During expansion, the rule's output may be modified given the presence of the modifier symbol.

Existing work shows it is certainly possible to successfully model rules which may be modified. An alternative flexible matching scheme (Siyari & Gallé, 2017) was shown to offer an improvement when identifying syntax within linguistic data, and to produce more compact encodings of both linguistic and symbolic representations of DNA sequences. The scheme itself was constrained to providing an alternative prefix and suffix for each flexible match on subsequent instances of a given rule, and as such relied on the explicit encoding of each variation to be included upon model expansion. A possible advantage of the transform-based method presented in this chapter is the potential for implicit operations, and the use of specific domain-based transformations, which may be applied by the program which expands the encoding, allowing for the inclusion of very complex and sophisticated modifications. In this manner, blends and overlaps between segements may also be made; however, this idea is beyond the scope of the current thesis, and as such is left for future work.

A key hypothesis for this study is that there is a relationship between the compactness of a model representing specific data, and the accuracy of the information which may be obtained from examination of that model. Assuming the existence of a deliberate hierarchical structure within the data, such as human-selected musical form or repetition, the Minimum Description Length principle (Rissanen, 1978) suggests that compression of that data by an algorithm which can generate such hierarchical structure is most likely to return a structure as close to the original as possible where the most compact possible model is generated, as opposed to larger or non-hierarchical models. In the case of a musical score, expected structure (Spring & Hutcheson, 2013) can include high-level concepts such as form (for example, Rondo, Fugue or Sonata), or medium-level organisational units such as phrases and measures. The ability to discover such structure as a result of processing by algorithm would mean it were possible to automatically obtain a structural analysis of a given musical score, such as may be performed by an expert musicologist, which would be helpful in a great many music-related tasks, such as understanding or teaching composition, grouping or selecting works by similarity, or the generation of entirely novel structures through comparison against a database of known pieces. Any method which contributes towards an automatic analytical approach, whether specific to a given school of analysis or universal in nature, is a valuable contribution.

As shown in Chapter 5, there is a general correlation between the effectiveness of the compressor used and its rate of success on any of the analysis-based experimental tasks; this may be seen particularly in the comparison between ZZ and IRR, where ZZ consistently shows stronger performance. If the input data contains terms which repeat within it, but with minor differences, it may be possible to achieve greater compression by considering all such instances relative to the "template" term, avoiding the costly need to add each as separate production rules.

This concept is strongly related to an additional hypothesis: where a sequence is modelled partly from patterns whose forms vary within it, those variations may be considered incidental, and the underlying similarity between pattern instances seen as the fundamental "template" upon which all such patterns are based. The manner in which an instance is transformed from its fundamental form is a specific representation of its variation, and basing substring similarity within the sequence primarily on the fundamental differences between patterns instead of their literal sequence allows a higher-level form to be identified. Very minor variations between patterns are clearly heard within musical sequences, in a change from major to minor for example, or from one mode to another. Capture of this transformed structure is likely to expose the patterns fundamental to the sequence, and, separately, the types of variation occurring within it. Where this simplifies the understood structure, greater compression should also be possible, alongside a more fundamental segmentation of the sequence itself.

6.1.1 Proposed Solution

It is possible to augment a grammar by allowing approximate matches to be made whilst seeking repeating terms within the input, and storing the information required to reconstruct each approximately-similar instance given the template term. The cost of signalling that a specific instance of a rule requires modification, and the cost of storing the instructions required to perform the correct modification, must be considered as part of the encoding of the grammar.

In this chapter, a solution is proposed to the problem of constructing a grammar which allows substrings to be flexibly matched, and yet is encoded with little difference to a straight-line grammar, allowing it to be processed and manipulated by methods which are designed to operate on such encodings. The solution makes use of a set of versatile transformations which may be customised to allow approximate equality between substrings using specialised domain knowledge, or basic symbolic or geometric operations such as insertions or reversal. The idea extends the work of Carrascosa et al. (2010), and follows the same generalised process of constructing a compact grammar G from an input string S:

1. Discover equivalent substrings in S,
2. Pick a combination of substrings to use in construction of G,
3. Construct a directed acyclic graph whose edges represent choosing or rejecting the replacement of a substring,
4. Find a minimal parse of the graph, and compute the grammar's length $|G|$ from both parse and substrings, and
5. Repeat the process from step 2 until no further reduction in $|G|$ is possible.

This approach represents an alternative to flexible matching schemes such as was proposed by Siyari and Gallé (2017), and is not limited to any specific definition of substring similarity.

6.2 Applications

Grammars which allow rule modification during expansion may benefit any application where grammar-based compression is appropriate, including general compression tasks where exact reproduction of the input is desired, and the nature of the data being processed means that such compression is more successful. Siyari and Gallé's flexible approach (Siyari & Gallé, 2017) was shown to produce smaller encodings for the Canterbury Corpus than that of Benz and Kötzing's GA-MMAS algorithm (2013). Where rule modifications allow a more inclusive modelling of patterns within the input data, more compact encodings than those attainable by grammar constructors such as ZZ or GA-MMAS may result, potentially even more compact than those produced by Siyari and Gallé's method.

Since the addition of transforms is an augmentation of straight-line grammars, a grammar constructor which allows for rule modification possesses all the advantages of the straight-line scheme, along with the benefits which transformations provide, discussed within this chapter. One particular advantage of such a grammar is its hierarchical structure. Such a structure may represent a hierarchy which underlies the input data, possibly chosen with purpose where the input is human-generated. A hierarchy may have creative implications, such as sub-divisions within the musical form of a classical piece, or a compositional meaning, as may be found in DNA strings by elements which are repeated within larger structural segments. Grammars which allow rule modification may also be applied to any application where discrimination of a hierarchy is useful, and a better-fitting hierarchy may be discovered where it includes segments which make use of transformed terms.

The following section describes the fundamental operations required by a grammar constructor which is based upon the exact matching of substrings, and the subsequent section presents a method by which such grammars may be augmented to allow the modelling of flexibly-matching substrings.

6.3 Exact Matching

6.3.1 Selection of Candidate Terms

To demonstrate the difference of approach between exact and flexible matching, consider the following process, which may be used to discover and represent exactly repeating segments during grammar construction. More efficient methods of repeat discovery exist; this explanation is included only for illustrative purposes.

Algorithm 1 Find all exactly repeating segments

```
Require: $S$, $minLength$, $maxLength$
Ensure: $D$, (dictionary of all exact repeats)
 1: for $l \leftarrow minLength$ to $maxLength$ do
 2:     for $i \leftarrow 1$ to $length(S) - l$ do
 3:         $a \leftarrow S[i : i + l]$
 4:         for $j \leftarrow 1$ to $length(S) - l$ do
 5:             $b \leftarrow S[j : j + l]$
 6:             if $a = b$ then
 7:                 if doesn't exist $D[a]$ then
 8:                     $D[a][1] = [i, l]$
 9:                 end if
10:                 $D[a][\text{end}] = j$
11:             end if
12:         end for
13:     end for
14: end for
```

The procedure results in a dictionary D with key set k, where $|k| = u$ and $0 \leq u \leq 5(length(S)+1)/2$. Each element of k represents a term t from S, and each term has a first occurrence at $i = D[t][1]$ with length $l = D[t][2]$, with subsequent occurrences at $D[t][\geq 3]$. All terms repeat at least once in S. The complexity of this procedure is $O(rn^2)$, where n is the length of S and r is the range of lengths to search for bounded by n to make $O(n^3)$. Each term may be considered a *constituent*; that is, it may be added to a grammar as a production rule, and its instances $D[t]$ in S replaced with a symbol which references it.

There are several alternatives to Algorithm 1. One possibility is the use of suffix trees (Stoye & Gusfield, 2002), where each node in the tree represents a symbol present in S, and repeating terms of n symbols in length may then be discovered by traversing from the root through any n nodes. This approach can have time complexity $O(n)$ (Ukkonen, 1995) for tree construction and term discovery. It is also possible to construct a matrix based on Levenshtein distance, using an algorithm such as that proposed by Wagner and Fischer (1974), and seek diagonals with unchanging values to identify the positions of repeating terms. This approach is less efficient at $O(n^2)$.

6.3.2 Constituent Selection and Minimal Grammar Parsing

For a typical input S, many constituents exist. Where there are i constituents, the search space of possible constituents to include in a grammar is 2^i. As shown by Carrascosa et al. (2011), it is possible to traverse this search space in $O(n^7)$, where n is the length of the input, by progressively adding or removing a constituent from the set of those currently selected, until a locally smallest grammar is found. Separation of the problem into constituent choice and minimal grammar construction allows the parsing of the input to become a problem of minimal parsing, which may be achieved by constructing a directed acyclic graph g of length $l = |g|$, where $l - 1$ is the number of symbols present in the string to parse, and a termination symbol is present at the end of g. Each edge taken in g adds either a symbol from the string, or a symbol referencing an instance of one of the currently active constituents. A minimal traversal of this graph yields the shortest possible symbol string, including references to constituent rules, which can expand to form the original string. The graphs of the constituents themselves can also be parsed, allowing them to contain sub-constituents which further reduce the model's encoded length.

6.4 Flexible Matching

6.4.1 Selection of Candidate Terms

Instead of seeking segments a, b which contain identical symbolic sequences, taking $a = b$ directly, it is possible to instead identify those which may be transformed by some function f given a parameter T which defines how the transformation is performed, allowing $a = f(T, b)$. Where a and b are exactly equal, no T is required, and its absence may be taken as $a = f(b)$ where f simply returns its input. The transformation T can be computed by a function t, and so obtained by $T = t(b, a)$, where t returns the operations required to transform $b \Rightarrow a$. In principle, the function t might be extended to itself return a function capable of transforming $b \Rightarrow a$, in which case the transform may be considered a good representation of P in the context of Kolmogorov Complexity (Kolmogorov, 1963) since it would encapsulate the transformation program. However, this study focusses on the concept of T as a parameter, and any other definition of t is left for future work.

If T is not constrained, it is possible for $a = f(T, b)$ for *any* substring b, although theoretically the encoding length of T should prevent all terms a and b for being considered similarly equivalent. Nonetheless, quantifying the degree of similarity is necessary to allow discrimination between substrings. Therefore, a cost c may be calculated for T, based on the instructions it contains, and any transform with $c > w$, where w is a threshold of reasonable cost, may be discarded. In this scheme, c may be said to represent the *cost* of altering the difference between a and b, and a measure of the complexity of describing T.

Where $|a| = |b|$, Algorithm 1 may be used to discover matching terms, replacing the comparison $a = b$ with $a = T(b)$. There is additional complexity to consider for this process, which depends on the specific algorithm used to create the function T, but this is not necessarily greater than a regular comparison; for instance, symbols between strings a, b may be compared by indices i, j in $O(n)$ time, regardless of whether j increments or decrements at each step. In the worst case, if $|a| \approx |b|$, the following process may be used to discover all terms for D:

The procedure results in a dictionary D as in Algorithm 1, but in this case an additional value is stored in $D[a][\geq 2]$ for each match to the template a, containing the length of the match (m in Algorithm 2). The complexity is increased to $O(mrn^2)$, where m and r are bounded by n to make $O(n^4)$. The cost of a flexible repeat a may be stored within D, or may be re-computed when required by querying the function t to recover the transform T. The cost of T may be taken as $|T|$ in some scheme γ, where γ may depend on the context of the model. In the case of a musical score, γ may be the space of possible compositional variations, pre-defining the context, and $|T|$ may be a minimal set of instructions to transform $a \Rightarrow b$.

Algorithm 2 Find all exact and flexible repeats

```
Require: $S$, $minLength$, $maxLength$, $w$
Ensure: $D$, (dictionary of all exact and approximate repeats)
 1: for $l \leftarrow minLength$ to $maxLength$ do
 2:     for $i \leftarrow 1$ to $length(S) - l$ do
 3:         $a \leftarrow S[i : i + l]$
 4:         for $m \leftarrow minLength$ to $maxLength$ do
 5:             for $j \leftarrow 1$ to $length(S) - m$ do
 6:                 $b \leftarrow S[j : j + m]$
 7:                 if $a \approx b|w$ then        ▷ If $a$ is approximately equal to $b$,
                                                     beneath some threshold $w$
 8:                     if doesn't exist $D[a]$ then
 9:                         $D[a][1] = [i, l]$
10:                     end if
11:                     $D[a][\text{ end }] = [j, m]$
12:                 end if
13:             end for
14:         end for
15:     end for
16: end for
```

It is of course possible to further reduce the complexity of the match discovery process. At minimum, a threshold w of reasonable cost may be set based upon the minimum gain required to make the match useful in the grammar construction process. Unfortunately, selection of an appropriate w is not a trivial calculation: for a function T which transforms $a \Rightarrow b$ and features as part of a smallest grammar, there may be only one instance of b within S since b is chosen as an approximate match for term a (and is likely to form the exact template for its own constituent if other instances of b exist elsewhere in S). However, the same transform T might also be applied to many other terms, and so storage of the encoded form of T and its separator with size $|T| + 1$ does not necessarily prevent T from being used to reduce the encoding size $|G|$, even when $|T| + 1$ is relatively large. Thus, treating w as a number of symbols and limiting it to the count required to store the term a it transforms could conceivably prevent a transform $|T| > w$ from being considered, and result in a larger than optimal grammar. A calculation of the maximum gain possible from T could be made, based on the terms $term_1, term_2 \ldots term_n$ it can be applied to, but this more challenging; it is not impossible that T may apply to an entirely new symbol combination, composed of original symbols from S and references added to grammar rules, which will not be seen until the construction process is underway, and thus its potential gain depends on how the grammar is minimised. It is reasonable to suggest that more use may be made of compact forms of T, perhaps relative to $|a|$ or $|b|$, and especially to those which will apply to a greater number of matches; as such, w may be set as a proportion of $|a|$ or $|b|$, and used as a rough parameter to constrain the space of transforms considered during the selection of candidate constituents.

6.4.2 Augmenting Grammars with Rule Transforms

To elaborate on the above, consider the following string, composed of a repeating term $a = [1, 2, 3, 4]$:

$S = [1, 2, 3, 4, 1, 2, 3, 4]$

Here, $|S| = 8$ symbols. This may be encoded in grammar form by adding a termination symbol \$, marking all preceeding symbols as belonging to the starting symbol S from $G =< \Sigma, N, S, P >$:

$G := 1, 2, 3, 4, 1, 2, 3, 4, \$$

This encoding results in $|G| = 9$ symbols. The term itself may now be added to G as the first production rule in P, separated from S with an additional termination symbol:

$G := r_1, r_1, \$, 1, 2, 3, 4, \$$

Now, $|G| = 8$ symbols, and represents a compressed version of S encoded as a grammar. It is not shorter than the original string, but no larger, despite the inclusion of two termination symbols \$ and two rule references r_n. Now consider a modified version of S, composed of the same term $a =$

$[1, 2, 3, 4]$ and a single reversed match $b = [4, 3, 2, 1]$:

$S = [1, 2, 3, 4, 4, 3, 2, 1]$

Again, $|S| = 8$ symbols. This may be encoded in grammar form as follows:

$G := 1, 2, 3, 4, 4, 3, 2, 1, \$$

The encoding results in $|G| = 9$ symbols. The term itself may be added to G as the first production rule in P, separated from S with an additional termination symbol, but it cannot yet provide any compression as it is not repeated in S:

$G := r_1, 4, 3, 2, 1, \$, 1, 2, 3, 4, \$$

This encoding is of size $|G| = 11$ symbols, and greater than its representation without r_1. We may mark symbols 5-8 as belonging to r_1, temporarily ignoring the need to reverse them upon expansion into the original string S:

$G := r_1, r_1, \$, 1, 2, 3, 4\$$

This model is compact at $|G| = 8$, and seems intuitively correct, but cannot yet reproduce the string it represents. We may mark that the second occurrence of r_1 requires modification, denoting this with the symbol $'$:

$G := r_1, r_1,', \$, 1, 2, 3, 4, \$$

Although now $|G| = 9$ symbols, it is only possible to obtain the original input if the prime is taken as a specific transformation, reversing any rule preceeding it. This is a valid encoding scheme, but limited by the lack of other available transforms, since $T = reverse(a)$ and therefore $r_n = a, r'_n = reverse(a)$ must be implicit. The addition of the modification symbol has not added significantly to the alphabet of the grammar; indeed, adding a second production rule instead would have increased it by the same amount. If two other symbols are chosen to represent the transform type, selecting $R = reverse(S)$ for this example, and a break point between the production rules r_n and transformation rules t_n, choosing !, we obtain the following encoding:

$G := r_1, r_1, t_1, \$, 1, 2, 3, 4, !, R, \$$

Here, $|G| = 11$, and so the representation, although seemingly reasonable, is unable to compress the original string S. In order to benefit any single existing rule r_n, a transform encoded in this manner must enable the removal of sufficient symbols from G to cancel out the cost of its inclusion in the grammar and the transform reference appended to a rule reference, amounting to $g = |T| + 1 + 1$. Given a constituent of length l, a gain of $l - 1$ is possible upon unmodified replacement of a single occurrence of the constituent's term, due to the addition of its reference. Thus, $gain > cost$ equalling $l - 1 > |T| + 2$ must be satisfied before any gain is possible relative to a single rule. In the above example, the potential gain is 3 symbols, at a cost of 3 for inclusion of T within the encoding, and so it is impossible to use T to produce a more compact form of G. Increasing $l = 5$ demonstrates this effect:

$G := 1, 2, 3, 4, 5, 5, 4, 3, 2, 1, \$$

The uncompressed encoding of this grammar has $|G| = 11$. Once again, a new production rule can be generated for the first occurrence of the term:

$G := r_1, 5, 4, 3, 2, 1, \$, 1, 2, 3, 4, 5, \$$

Not surprisingly, adding the rule, which requires additional terminator and reference symbols, does not produce a compressed model; now, $|G| = 13$. However, since $l = 5$, it is now possible to produce a smaller encoding than that shown above:

$G := r_1, r_1, t_1, \$, 1, 2, 3, 4, 5, !, R, \$$

The inclusion of T here results in $|G| = 12$, which, although not yet a reduction in encoded size, is nonetheless a smaller and potentially "better" explanation than the above grammar without transforms. It is also now possible to denote new transform types, by adding new symbols which modify the preceeding rule in a specific manner.

With this scheme, where generally $|T| \ll l$ and T may be applied to $\gg 1$ rules, compression greater than an encoding which does not include transforms is possible.

6.4.3 Modifier and Constituent Relationships to Grammar Size

As a simple demonstration of the relationships between modifier and constituent attributes, a constrained grammar scenario may be modelled, with the following assumptions:

1. Input data consists only of n repeating terms.
2. Only one transform T, of length t, is present within the final grammar.
3. Each constituent is associated with o exactly matching substrings within the input.
4. Each constituent has an identical length of l symbols.
5. The transform T applies to exactly one instance of each constituent.
6. The substrings represented by all constituents, and all terms which may be transformed by T to match any constituent's substring, are distinct from each other.

The variables n, t, o, l may now be assigned different values, and the metrics $|G_{input}|, |G_{std}|, |G_{mod}|$ calculated for encoded input length, standard grammar length, and grammar length when T is included, respectively. These equations may be defined as follows:

$$|G_{input}| = n(lo + l) + 1 \tag{6.1}$$

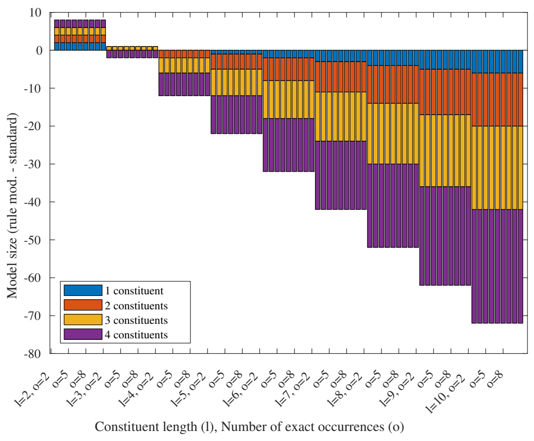
$$|G_{std}| = n(o + l) + n(l + 1) + 1 \tag{6.2}$$

$$|G_{std}| = n(o + 2) + n(l + 1) + t + 2 \tag{6.3}$$

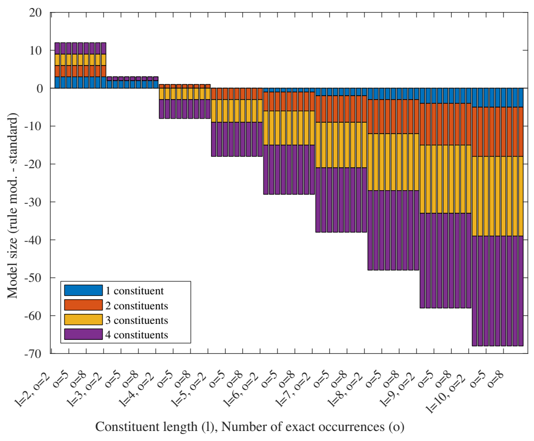
Given that all terms, including those which may be transformed by T, are distinct, it is possible to state that a grammar which does not make use T will instead possess an additional l symbols, for the moment assigned to the label x, as these occur only once within the input and are not matched by any other constituent. Within a grammar including T, these symbols may be replaced by exactly one reference to the constituent with term c for which $x = T(c)$, and one reference to the transform T which modifies it. An additional separator symbol is required within the encoding between the existing symbols and the encoded transform. Although the scenario is clearly heavily constrained, it does allow an evaluation of the effects of varying transform lengths, constituent lengths, and number of exact occurrences within the input to be made, with the aim of exposing some of the behaviour of grammars which include transforms under differing conditions.

Figure 6.1 charts the difference in grammar sizes given various values for n, o, l where t is held constant. In general, it is clear that the more rules a transform applies to under these constraints, the greater the reduction in encoding length which may be achieved. However, it is only when the length of the substring each constituent represents is notably larger than the length of T that use of the transform becomes beneficial. The number of occurrences of each constituent, o, is unimportant to the overall gain achieved through the addition of the transform, since each exact occurrence is not applicable to T; that is, $T(c_a) \neq c_b$ for any pair of constituents a, b. Under real-world conditions, it is entirely possible that a given transform may apply to multiple, identical substring occurrences, and may therefore be useful in removing the necessity for an additional production rule to reduce encoding size by replacing them.

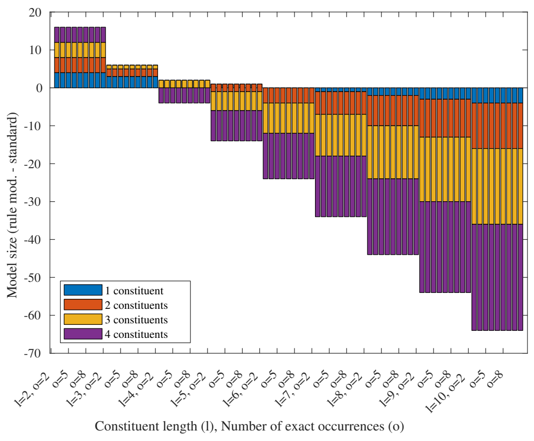
These figures show that rule modifiers can remain beneficial to the minimisation of $|G|$ even as l increases and only one instance of T is made per constituent, and in a worst-case scenario are unaffected by changes in o, providing each transform applies to a sufficient number of constituents to reduce encoding length, and are themselves much shorter in length than the constituents which they apply to.



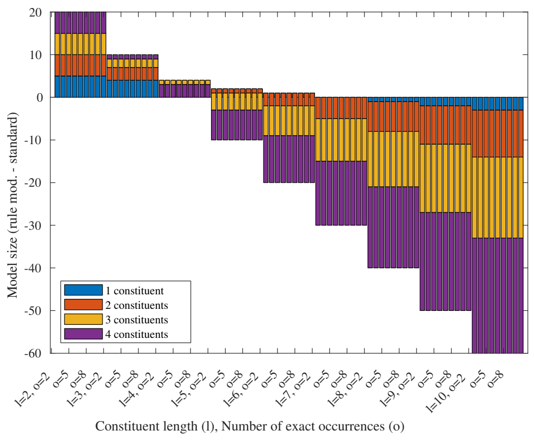
(a) *t = 1*



(b) *t = 2*



(c) *t = 3*



(d) *t = 4*

Figure 6.1: Difference in output encoding size between grammars with and without the inclusion of transform T, for transform lengths $t = 1, 2, 3, 4$. Overlaid plots are shown for inputs containing $n = 1 : 4$ constituents, with results shown for constituents of length $l = 1 : 10$, and exact occurrences $o = 1 : 10$ for each length.

6.4.4 Transform Encoding

As shown, a grammar G may be encoded as a sequence S, P, Z, extending its definition to include the encoded set of all $t \in Z$ which apply to S, P. Where n rules are present in P, references $r_{n+1}, r_{n+2} \ldots r_{n+j}$ can refer to transforms $t_i, t_{i+1} \ldots t_{i+j}$, which requires the addition of a single unique symbol to the alphabet to mark the separation between P and Z. For this study, each type of transform is represented by a unique symbol in the same manner as non-terminal rule references r_n, and any additional information required to complete the transformation of $a \Rightarrow b$ is added sequentially to t_n prior to the termination symbol.

To allow a comparison to be made between the properties of grammars constructed from differing input data, and across different applications, a limited set of transforms were chosen for this study, composed of operations which may apply to generic, primitive data, operations which may apply specifically to musical data, and those which may be appropriate to both types.

A Levenshtein edit-distance (1966) based modifier was added as a generic operation, as this allows generalised matching to occur between a pair of non-equivalent substrings, such as may be found in linguistic data, but matches are constrained by the minimum number of characters which must be edited to transform any term within the pair into the remaining term (Navarro, 2001). Although capable of calculating the number of symbol-level changes required to change one term into another, this form of edit distance measurement does not capture operations which apply at a term level, such as a reversal of sequence ordering. For example, to transform the word *rat* into *tar* at a symbolic level would require two substitutions: the replacement of *r* with *t*, and *t* with *r*. These are complementary changes, and a human would intuitively know a simple reversal of the entire sequence would produce the required output. Such a reversal also applies to musical data, where *retrograde* is a reversal in the order of a motif's note sequence (Spring & Hutcheson, 2013), and so the possibility of directly reversing a substring sequence is also included in the set of this study's transforms.

Three further transforms were added to the set, applying with increasing relevancy to musical data. The first of these is a simple numeric translation, where a set positive or negative value is added to every element in a sequence, providing the opportunity of matching one pattern to another of identical shape but with a relative offset. When working directly with pitch values, this allows simple transpositions of a musical pattern to occur (Spring & Hutcheson, 2013). The second transform attempts to capture more complex pattern manipulation by allowing the application of a "origin" value to a sequence, around which the elements of that sequence are "reflected". Where the origin value is outside the range of the values within a sequence, this will result in its inversion, which may then be transposed to produce a directly inverted motif, commonly used as a compositional device (Spring

& Hutcheson, 2013). Finally, transformation of a symbol sequence from an expected Western mode into a set of scale degrees, followed by expression of the pattern in an alternative mode, provides an attempt to capture a composer's use of mode change when repeating motives (Spring & Hutcheson, 2013). Although not representing an exhaustive exploration of the compositional space, the chosen modifiers are intended to enable the evaluation of a small range of generalised and domain-specific transforms when constructing grammars which allow rule modification, as a proof of concept upon which future work can be based.

The following provides a detailed explanation of the operation of each transform:

1. *Levenshtein:* A sequence of operations is generated by following the path through the Levenshtein matrix comparing two terms. Three operations are encoded: *substitute*, *insert* and *delete*. Each refers to a specific index in the sequence a which T modifies as $b = f(T, a)$, and two types of encoding are possible: *a)* an index may be associated with each operation, or *b)* each index in T may correspond with an index in a. The former is most suitable for modification of a large sequence a requiring few non-contiguous operations, the latter where many operations are required relative to the length of a.

   Where *(b)* is used, a fourth special symbol is required, *no-op*, which shows that no alteration of a_i is made where *no-op* occurs at index i. The operations *substitute* and *insert* require an additional parameter p, representing the symbol which should be substituted or inserted at the current index. For example, a Levenshtein transform for $a \Rightarrow b$ where $a = [1, 2, 3, 4, 5]$ and $b = [1, 6, 3, 5]$ might be encoded in form *(b)* as the sequence $T = [Levenshtein, no-op, subtitute, 6, no-op, delete]$, with length $|T| = 6$. In order for T to enable a reduction in $|G|$, it must also apply to other pairs of a, b, for example $a = [7, 8, 9, 0]$, $b = [7, 6, 9]$ or $a = [3, 3, 3, 3, 3, 3]$, $b = [3, 6, 3, 3, 3]$. Since either of the forms *(a)* and *(b)* are likely to cause $|T| \gg 1$, representing a Levenshtein-derived transform in this manner mandates it should apply to $n \gg 1$ constituents in G to be included to reduce $|G|$.

2. *Reverse:* Where $a = s[i : j]$ for some string s and there exists an equivalence $b = s[j : i]$, a simple transform T may be encoded as $T = [reverse]$. Since $|T| = 1$, minimal occurrences of $b = f(T, a)$ need to exist within G for T to offer a potential reduction in $|G|$, providing such equivalences are present.

   This transform may be used as a simplistic model of a musical "retrograde" (Apel, 2003), in which a group of notes or phrase is scored with an exact reversal of its pitch sequence and rhythm.

3. *Translate:* If element-wise addition is applied to a resulting in $b = a + d$, where d is the difference in indices between a and b for all elements, then a may be translated in alphabetic space by T to give $b = f(T, a)$. It is only necessary to specify d, and so a transform T may be encoded as $T = [translate, d]$. There is only a single symbol of difference in length between $T_{reverse}$ and $T_{translate}$, and so the transform provides a reasonable opportunity for reducing $|G|$

with $|T| = 2$. However, unlike $T_{reverse}$ which may be applied to any suitable a, b pair, there exist $2c - 1$ possible combinations of $T_{translate}$ where c is the size of the terminal alphabet derived from the original input string G is constructed from. Therefore, it is necessary for there to be several instances of $b = f(T, a)$ for each $T_{translate}$ chosen for inclusion in G which reduces $|G|$.

In the context of musical score data, where the alphabet is composed of chromatic pitches, a translation may replace the function of data conversion to intervals, which is commonly performed to provide transposition invariance by increasing equivalence. As discussed by Cambouropolous et al. (Cambouropoulos et al., 2001), such conversion introduces the undesirable property of dual membership of a single note to two contiguous and adjacent rules. The use of a transform to translate patterns composed of chromatic pitches also provides increased equivalence, but avoids this issue, and represents an online alternative to pre-processing into intervals.

4. *Reflection around an axis:* A term a may be equivalent to a term b which has been rotated on the horizontal axis whose origin is offset vertically. Given this offset o in alphabetic space, an element-wise operation is applied to a resulting in $b = a - o - 2(a - o) + o$. All that is necessary to perform this operation for any term a is the value of o, thus a transform may be encoded as $T = [axis, o]$. Like the *Translate* transform, encoding length is short, giving a reasonable chance of the transform reducing $|G|$ if a few instances of reflection exist within the input.

   This transform may be used as a simplistic version of musical reflection, or horizontal mirroring (Kempf, 1996). Mirroring occurs in-place and without transposition only when $o = \min a + \max a - \min a/2$. For all other values of o, an offset is introduced, for which additional translation may be required to produce equality between a and all potential b.

5. *Mode:* This transform is based on the concept of modes in Western music, where there exists a vector of $n - 1$ intervals, each representing the difference in semitones between the n degrees of the scale for that mode. For instance, Ionian has chromatic intervals $[2, 2, 1, 2, 2, 2, 1]$, and Dorian has $[2, 1, 2, 2, 2, 1, 2]$, a left rotation of Ionian. Any note within a musical score may be separated into the components *root*, *octave*, *mode*, *degree*, which may also be represented as *offset*, *mode*, *degree* where $offset = 12octave + root$. A transform between mode for the pair a, b may be simply made where the following conditions are satisfied: *(a)* a and b are chromatic pitch values, *(b)* $a_{offset} = b_{offset}$, and *(c)* both terms have identical degrees for two different modes m_a, m_b. Where these are true, a may be reduced to scale degrees by $offset = \min a$ and $a_{degrees} = a - offset$. As set of possible modes my be obtained by eliminating any candidates which do not contain chromatic degrees present in $a_{degrees}$ (such as 1, 3, 6, 8, 10 in Ionian), and a transform T may be constructed for a, b where $b_{degrees} = b - offset$ and $mode(a_{degrees}, m_a) = mode(b_{degrees}, m_b)$.

   Using this process, the information required for a transform $a \Rightarrow b$ is relatively minimal, as $b = f(T, a)$ can be completed as long as m_a and m_b are known. Thus, T may be encoded as $T = [mode, m_a, m_b]$. For a small alphabet in a, b there may be up to $v^2 - v$ combinations of m_a, m_b where v is the number of available modes, and since optimal transform use is ensured when T applies to the most possible rules in G it is desirable to consider all possibilities for $m_a, m_b, m_a \neq m_b$ when constructing G. This can strongly increase the size of the search space of all T. In this study, only the seven "modern modes" are considered, comprised of Ionian to Locrian as all possible rotations of the interval sequence $[2, 2, 1, 2, 2, 2, 1]$.

This method allow for the addition of *any* transformation which is compact enough in its encoding length l to allow positive gain g over all instances of its use across all production rules, and thus satisfying the condition $l + 1 < g$. There is great versatility in the types of possible transform, which may range from generic operations which apply to a wide range of data, to domain-dependent operations crafted specifically to represent structural or element-based constructs to be found in the input. Theoretically, all possible transformations could be considered during substring search and grammar construction.

This differs significantly to some existing flexible matching schemes, such as that proposed by Siyari and Gallé (2017) which constrains the search space to terms with the same prefix and suffix. However, note that an approximation of their scheme is possible in this method through the use of "Levenshtein" transforms, if the encoding is such that a string of symbols with no fixed length may follow an instruction to substitute a segment of a rule which is bounded by a start and end index. For instance, to model equivalence between the terms $a = [1, 2, 3, 4]$, $b = [1, 5, 6, 4]$ and $c = [1, 7, 8, 4]$, two transforms with the form $T = [Levenshtein, index, s_1, s_2 \ldots s_n]$ may be encoded as $T_1 = [Levenshtein, 1, 5, 6]$ and $T_2 = [Levenshtein, 1, 7, 8]$. Providing the form of any given T is defined at the time of encoding and decoding, any suitable form may be chosen.

6.4.5 Compound Transforms

For any substring term $s_1, s_2 \ldots s_n$, a match between it and another term may depend on the application of multiple transforms. For instance, it is quite possible that a given musical phrase might exist elsewhere within a score in a reversed form, but transposed into an alternative key, requiring both transformations to be applied before $a \Rightarrow b$ is complete. In this case, a solution could be to specify both transforms sequentially following the rule reference within G, and apply each during expansion of that rule. Where the order the transforms are applied in is important, they may simply be specified in that order within the encoding. For instance, where $a = [1, 2, 3, 4]$ and $b = [4, 3, 3, 1]$, a reversal T_1 of a is required, but the symbol 2 must also be replaced by T_2. If this is done pre-reversal then index 1 should be specified, and conversely index 2 is needed if the substitution is made post-reversal. The optimal selection of T_2 depends on its use elsewhere within the encoding, and also on any additional reduction in $|G|$ possible because of the sequence r_n, T_u, T_v. Potentially, T_u may apply to r_n in the absence of T_v in other places within G (perhaps the phrase a repeats most often without transposition, for instance), making the ordering r_n, T_u most suitable for a model of a particular score.

One approach to the representation of compound transforms when parsing a graph of G, is to allow an edge representing a constituent instance r_n of length l from position s requiring modification to lead to an intermediate vertex, instead of returning directly to the vertex representing position $s + l$ and associating a weight equivalent to the cost of the non-terminal for r_n plus its associated transforms T_u, T_v. Under such a scheme, the edge belonging to r_n may carry a weight of 1 to account for its non-terminal symbol, and additional edges may lead from its vertex through an arbitrary number of edges and vertices, each representing an applicable transform T_n and also with an associated edge weight of 1, to account for that transform's non-terminal symbol. During parsing, the constituent cannot be chosen without also selecting a return path through the transform vertices to $s + l$, at a cost of $1 + n_v$ where n_v is the count of transform vertices visited. The disadvantages of such an approach are twofold: each selected transform incurs the cost of an additional non-terminal to represent it within the encoded output, and overall graph complexity increases due to the extra vertices which must be included within it. Parsing of the graph alone cannot help discriminate between each T_n in selection of an ideal set of modifiers, just as it cannot select an ideal constituent set, only the shortest encoding possible given the available components – an optimisation approach such as ZZ is still necessary.

It is equally possible to specify compound transformations as unique, single transforms. An advantage of this approach is the ability to minimise the encoding of compound transforms to just their specific parameters. In the example above, two separate encodings are made: $T_1 = [reverse]$ and $T_2 = [Levenshtein, substitute, index, 3]$, each requiring a terminator symbol, resulting in $|G| + 7$. However, where it is known that an encoding may benefit from multiple applications of both transforms in this order, a single compound version $T = [reverseandLevenshtein, substitute, index, 3]$ may be stored instead, resulting in $|G| + 5 < |G| + 7$. Where the sequence is not known, all possible orderings may be considered, if the accompanying increase in search complexity is warranted. The overhead required to add such a compound transform to the encoding of G is one additional character in the alphabet, per transform. However, in the worst case, the use of such transforms causes a strong increase in the transform search space, and potentially, therefore, of the substring search space for each constituent.

To illustrate this, we can consider a simple example. Assume $n = 2$ transform types are to be used, both of which accept a single parameter with a range of three discrete values, yielding a parameter space of $p = 3$ possibilities per transform type. From this combination, $np = 6$ individual transforms exist which would each need to be considered during a search of the transform space. However, where the compound versions of these transforms are also to be considered, all possible combinations of possibilities and ordering for both transforms must be added to the space. In this case, each compound transform type requires p^n transforms to combine with its counterpart, and there are $n! \div (n - r)!$ permutations without repetition of transform types, with $r = n$ since there are only 2 types to consider. This results in a total of $p^n * n! = 18$ additional transforms for consideration, causing an increase from 6 to 24, even in this basic example. Where compound transforms are used, the number of transforms in the worst case may be defined as follows, assuming an equal parameter space p for each type:

$$\sum_{i=1}^{n} p^i \frac{n!}{(n-i)!}$$

Clearly, this results in a far greater transform search space, and may cause a significant increase in parse graph and substring search complexity depending on the character of the input string. Heuristics might be used to abort a search attempt early where a specific combination of transform types is known to be unsuitable, which can aid in constraining computational complexity.

6.5 Construction of a Flexible Grammar

The approach of Carrascosa et al. (2010; 2011; 2012) demonstrated that when attempting to solve the Smallest Grammar Problem it is possible to separate construction of a compact grammar into two sub-problems: constituent selection and Minimal Grammar Parsing. When considering the Smallest Grammar Problem, there is a direct dependency between the replacements which may be made within the input string and the substrings which have been selected as constituents for inclusion in the grammar, with respect to the minimal sequence length for the encoding G; inclusion of all constituents in the encoding results in a minimal-length representation of the input, but a maximal-length encoding of constituents. For a search space of c constituents, there are $O(2^c)$ combinations within the lattice which must be tested to minimise $|G|$, for which it is necessary to minimise both sub-domains. ZZ may be used as an optimiser to traverse the constituent search space, altering the $O(2^c)$ bound to $O(n^7)$ where n is the length of the input in symbols. ZZ explores the space of all possible constituents greedily, and a minimal encoding G is generated at each step given the current combination of constituents ZZ has selected. Traversal completes when no further improvement is possible when moving to a neighbouring node in the lattice.

Equally, there is a direct dependency between the replacements which may be made within the input string and the transformations which are available and potentially applicable to a subset of replacements. A smaller encoding may potentially result where a transform can be applied, as it can allow a constituent which does not exactly match the term being examined to be used as a replacement at a cost of a single symbol marking its modification. However, the inclusion of a transform adds to the length of the encoding, and, in a similar manner as inclusion of constituents, inclusion of all possible transforms results in a minimal-length representation of the input, but a maximal-length encoding of transforms. Given a particular constituent set C and an empty set of transforms, Minimal Grammar Parsing may be used to produce the shortest encoding possible for C. Where t possible transforms may be applied to C to increase the substrings which may be replaced, there are $O(2^t)$ combinations within the lattice of transforms which must be tested to minimise $|G|$ given C. Thus, adding transforms to a grammar also adds a dimension to the lattice, which increases in size to 2^{ct} nodes.

6.5.1 Traversal of the Search Space

ZZ may be extended to provide a means of greedily selecting a combination of both constituents and transforms. For each constituent combination C and transform combination $\mathcal{M}$, all possible additions of a new constituent to C may be tested, and a new combination of C selected where $|G|$ is reduced. Following this step, all possible additions of a new transform t to $\mathcal{M}$ may also be tested, and a new combination of $\mathcal{M}$ selected where $|G||C$ is also reduced. In this manner, traversal through the space can continue, removing selections from C or $\mathcal{M}$ as appropriate following the ZZ algorithm until a locally minimal $|G|$ is discovered. At this point, traversal may stop, and a compact G given $C, \mathcal{M}$ output.

However, it is likely that addition of a new constituent c to C also provides opportunities for transforms which are applicable to $C \cup c$, but not $C \setminus c$. In this scenario, it is also possible that a number of existing transforms $t \in \mathcal{M}$ will no longer be useful given $C \cup c$, and will increase the size of the encoding beyond the minimum possible with a subset of $\mathcal{M} \setminus t$. Since a transform has no direct dependency to any single constituent, the optimal combination for $\mathcal{M}$ depends specifically on C and may not necessarily be found by simultaneously performing a stepwise traversal of the constituent and transform spaces.

As an example of this, consider an input $G = a, b, c, d, a, b, c, d, d, c, b, a, e, f, g, h, e, f, g, h, h, g, f, e, \$$, with an encoded length $|G| = 25$ symbols. Where a stepwise selection of C and $\mathcal{M}$ is conducted, all available constituents will first be considered for addition to C, with the candidate producing the greatest reduction in encoding length retained. Next, all available transforms will be considered for addition to $\mathcal{M}$, and, once again, the candidate which allows the greatest possible reduction in $|G|$ is added. Construction continues in this manner until no further reduction of $|G|$ is possible, at which point each $c \in C$ is removed in turn, followed by the removal of each $t \in \mathcal{M}$, in an attempt to reduce $|G|$. If the attempt succeeds, the process repeats with the current $C, \mathcal{M}$. When no further minimisation of $|G|$ is possible, the algorithm terminates and the current encoding is output as the smallest identified grammar. Given the input, and a candidate ordering of $C_{cand} = \{[a, b, c, d], [e, f, g, h]\}$, $M_{cand} = \{[translate, 4], [reverse]\}$, construction of G may proceed as follows:

1. The constituent sets $C \cup \{[a, b, c, d]\}$ and $C \cup \{[e, f, g, h]\}$ are evaluated, with $M = \emptyset$. Both produce a reduction of $|G| = 24$, and so the first candidate minimising $|G|$ is chosen. With $C = \{[a, b, c, d]\}, M = \emptyset$, the encoding $G = r_1, r_1, d, c, b, a, e, f, g, h, e, f, g, h, h, g, f, e, \$, a, b, c, d, \$$ results from parsing.

2. The modifier sets $M \cup \{[translate, 4]\}$ and $M \cup \{[reverse]\}$ are evaluated, with $C = \{[a, b, c, d]\}$. The first transform produces $|G| = 23$, the second $|G| = 24$, and so $M = \{[translate, 4]\}$ is chosen, since it allows the existing production rule $r_1 = a, b, c, d$ to match the substring e, f, g, h within G, for two occurrences – simply reversing r_1 only allows a single occurence to be additionally matched. The encoding $G = r_1, r_1, d, c, b, a, r_1, t_1, r_1, t_1, h, g, f, e,$ $!, translate, 4, \$$ results from parsing, with the symbol ! indicating the start of transform encoding.

3. The remaining candidate consitituent is now tested, and $C \cup \{[e, f, g, h]\}$ is evaluated. By including the candidate, encoding length is increased to $|G| = 24$, since the modification of r_1 can now be stored as a new rule $r_2 = r_1, t_1$, saving two symbols, but two non-terminals for r_2 must be added along with a terminator for the rule, resulting in a loss of three symbols, giving $|G_{curr}| = |G_{prev}| + 1$ overall. Since $G = r_1, r_1, d, c, b, a, r_2, r_2, h, g, f, e, \$, a, b, c, d, \$, r_1, t_1,$ $!, translate, 4\$$ is a larger encoding, $[e, f, g, h]$ is rejected for inclusion in C.

4. The remaining transform is tested next, and $M \cup \{[reverse]\}$ is evaluated. Assuming compound modifiers are allowed, the new modification allows the expansion of both the production rule a, b, c, d and its modified version e, f, g, h to match both reversed instances of those terms within G, resulting in a reduced $|G| = 22$ with the encoding $G = r_1, r_1, r_1, t_2, r_1, t_1, r_1,$ $t_1, r_1, t_2, t_1, \$, a, b, c, d, !, translate, 4, \$, reverse, \$$. The transform is retained in M.

5. The remaining candidate is now tested, and $C \cup \{[e, f, g, h]\}$ is evaluated. It is now possible to replace all occurrences of the candidate in G with instances of a new production rule r_2, and assign it to be a modified version of r_1, as $r_2 = r_1, t_1$. This saves 3 symbols by removing the need to follow r_1 with t_1 in three instances, but adds an overhead of 3 symbols in storing r_2, producing an encoding $G = r_1, r_1, r_1, t_2, r_2, r_2, r_2, t_2, \$, a, b, c, d, \$, r_1, t_1, !, translate, 4, \$, reverse, \$$ with length $|G| = 22$. Since the candidate does not reduce $|G|$, it is rejected for inclusion in C.

6. Since all candidates have been tested and no smaller $|G|$ could be found, members of $C = \{[a, b, c, d]\}$ and $M = \{[translate, 4], [reverse]\}$ are now removed in turn, to discover which, if any, are preventing the generation of a smaller encoding. Removing the single candidate from C also removes the possibility of using any transform in M, and is not constructive. Choosing $M = M - \{[translate, 4]\}$ means r_1 can no longer match any substring based on e, f, g, h,

and increases encoding length to $|G| = 24$. Choosing $M = M - \{[reverse]\}$ means reversed versions of either candidate cannot be represented by any rule since they occur only once in the grammar, and this increases encoding length to $|G| = 23$.

7. Stepwise addition or removal of any single element given $C = \{[a, b, c, d]\}, M = \{[translate, 4], [reverse]\}$ has proven unsuccessful, and so the algorithm terminates, and the encoding $G = r_1, r_1, r_1, t_2, r_1, t_1, r_1, t_1, r_1, t_2, t_1, \$, a, b, c, d, !, translate, 4, \$, reverse, \$$ is output with $|G| = 22$.

Unfortunately, a smaller encoding does exist: if the two substrings a, b, c, d and e, f, g, h are seen as separate entities, which exist with both repeated and reversed instances within the input, it is possible to generate an encoding $G = r_1, r_1, r_1, t_1, r_2, r_2, r_2, t_1, \$, a, b, c, d, \$, e, f, g, h, !, R, \$$ with length $|G| = 21$. This combination for C, M could not be reached in a stepwise fashion because the transform *translate*, 4 offered an improvement when added, but its presence in the encoding prevented the constituent e, f, g, h from reducing $|G|$, and so the latter was not available when the optimal transform *reverse* was later considered.

Although overly simplistic, this example helps to demonstrate why it is not possible to reach a globally-optimal state when treating constituents and rule modifiers as entirely separate entities which may be independently selected. Instead, given a set of constituents C, there exists an optimal selection of modifiers M composed only of transforms which minimise $|G|$. As such, if C_{min} represents a set of constituents which produce a minimal encoding given a set of candidate constituents C_{cand}, it is not possible to derive M_{min} from M_{cand} without first knowing the set C_{min} to which the modifications will apply, or vice versa – no direct relationship exists between a transform and a constituent, but a set of ideal transforms is entirely dependent upon its associated set of constituents, just as a set of ideal constituents is entirely dependent upon its associated set of transforms.

A stepwise approach to solving this problem is clearly not optimal. Instead, an *entirely separate* ZZ traversal of the constituent space may be made for each move ZZ makes between nodes of the transform space. Although this clearly introduces greater computational complexity to the attempt to minimise $|G|$, the result is consideration of all combinations of C returned by a local ZZ search, and the possibility of measuring their effect for a given transform combination M, thus minimising $|G|$ by manipulating C given M. In this manner, C is optimised for each M_{curr} considered, and both searches of the lattice are conducted independently to produce locally optimal results.

The following example provides a simplified demonstration of the approach outlined above, beginning again with $G = a, b, c, d, a, b, c, d, d, c, b, a, e, f, g, h, e, f, g, h, h, g, f, e, \$$:

1. To begin with, $C = \emptyset$ and $M = \emptyset$. For each candidate constituent c_i, the constituent set $C \cup c_i$ is evaluated, resulting in two grammars of identical encoding length $|G| = 24$, since

both candidates replace exactly 8 symbols each, and require 2 non-terminals and a production rule of encoded length 5. The first candidate which reduces $|G|$ is retained, resulting in $C = \{[a, b, c, d]\}$.

2. Next, any remaining constituents are tested, resulting in an evaluation being made for $C \cup \{[e, f, g, h,]\}$. As with the first constituent, a reduction of 1 symbol is possible, and so with no remaining candidates to consider an encoding of length $|G| = 23$ is returned. No further improvement is possible through removal of any element of C, and so the encoding $G = r_1, r_1, d, c, b, a, r_2, r_2, h, g, f, e, \$, a, b, c, d, \$, e, f, g, h, \$$ is recorded as the most compact encoding seen given $\mathcal{M} = \emptyset$. This encoding forms the baseline against which all models with $\mathcal{M} \neq \emptyset$ will be compared.

3. A vertical movement is now made through the lattice of transforms, producing $\mathcal{M} = \{[translate, 4]\}$. Starting again with $C = \emptyset$, each candidate constituent c_i is added to C, resulting in two encodings of lengths $|G| = 23$ and $|G| = 27$ for the candidates a, b, c, d and e, f, g, h respectively, since the modifier in $\mathcal{M}$ can only transform the former candidate string into the latter. $C = \{[a, b, c, d]\}$ is therefore retained.

4. The remaining constituents are tested, resulting in an evaluation of $C \cup \{[e, f, g, h]\}$. This generates an encoding of length $|G| = 24$, since the transform *translate*, 4 can be used to shorten the encoding of the candidate production rule. With no further constituents to test in this simple example, and no improvement possible through the removal of any element of C, the encoding $G = r_1, r_1, d, c, b, a, r_2, r_2, h, g, f, e, \$, a, b, c, d, \$, r_1, t_1, !, translate, 4, \$$ is recorded as the most compact encoding seen given $\mathcal{M} = \{[translate, 4]\}$.

5. A further candidate to be considered for initial addition to $\mathcal{M}$ remains, and so the set of rule modifiers becomes $\mathcal{M} = \{[reverse]\}$. Starting once more with $C = \emptyset$, each candidate constituent is added individually to C, producing two encodings of identical length $|G| = 24$, since the transform *reverse* applies equally to both constituents. The first candidate which reduces $|G|$ is retained, resulting in $C = \{[a, b, c, d]\}$.

6. The remaining constituents are now tested, resulting in the evaluation of $C \cup \{[e, f, g, h]\}$. This generates the ideal encoding of length $|G| = 21$, and, since there are no more candidates to consider and no removal of elements of C can generate a smaller model, the encoding $G = r_1, r_1, r_1, t_1, r_2, r_2, r_2, t_1, \$, a, b, c, d, \$, e, f, g, h, !, reverse, \$$ is recorded as the most compact seen given $\mathcal{M} = \{[reverse]\}$.

7. Having exhaustively tested all single-transform additions to $\mathcal{M}$, it is now possible to select a candidate for retention as the search of the transform space continues. There are two possibilities, with encodings of length $|G| = 23$ and $|G| = 21$ for the first and second transforms respectively. As such, the first is retained, and the search continues from the node $\mathcal{M} = \{[reverse]\}$ in the lattice.

8. A vertical movement can now be made through transform space, producing $M = \{[translate, 4], [reverse]\}$. Starting with $C = \emptyset$, each candidate is considered for addition to C, resulting in encodings of length $|G| = 22$ and $|G| = 27$ for the two respective candidates. The first addition is retained, and so the set of constituents becomes $C = \{[a, b, c, d]\}$.

9. The remaining constituents are tested once more, producing an evaluation for $C \cup \{[e, f, g, h]\}$, for which an encoding of length $|G| = 22$ results. There are no other candidates to consider and no improvement to be made through removal of elements in C, and since this encoding is not more compact than any previously encountered with the modifier set under test, the encoding $G = r_1, r_1, r_1, t_1, r_1, t_2, r_1, t_2, r_1, t_2, t_1, \$, a, b, c, d, !, translate, 4, \$, reverse, \$$ is recorded as the most compact encoding seen given $M = \{[translate, 4], [reverse]\}$.

10. Having exhaustively tested all single-transform additions to $M = \{[reverse]\}$, and discovered an encoding of length $|G| = 22$ which is *not* more compact than that observed for $M = \{[reverse]\}$ alone, any additions to M are rejected. Only the removal of individual modifiers may offer any opportunity for a reduced $|G|$. In this simple example, both possibilities – $M = \{[translate, 4]\}$ and $M = \{[reverse]\}$ – have already been observed, and cannot further refine the model.

11. Finally, with no movement possible in the C or M lattices which further reduces $|G|$, traversal ends. The ideal encoding $G = r_1, r_1, r_1, t_1, r_2, r_2, r_2, t_1, \$, a, b, c, d, \$, e, f, g, h, !, reverse, \$$ of length $|G| = 21$ is output.

Exploration of the smallest grammar which may be constructed given any specific combination of transforms M, alongside a ZZ traversal of the space of those combinations, have prevented any constituent combination from blocking the use of a particular transform. Thus, separation of the optimisation process into individual searches of constituent and transform space enables the discovery of an encoding which is locally optimal in both domains combined.

6.5.2 Reduction of Complexity using Heuristics

Without optimisation, traversing the lattice of all possible transform combinations for each chosen combination of constituents is an overly complex solution. Instead, heuristics may be employed to reduce the total number of nodes evaluated in both the transform and constituent dimensions.

Early Stopping

A ZZ traversal through the lattice of constituents is characterised by a decrease in encoding size for each node selected from the pool of neighbouring candidates. For a given node x, representing a combination of constituents C, all neighbours which add one constituent to C will be evaluated in a decending step, and all neighbours which remove one constituent will be evaluated in an ascending step. The size of the encoding for each, $|G|$, will vary, and three specific conditions are probable:

1. The range of values will be $< |G|$ to $> |G|$,
2. Variance within the values will be small relative to $|G|$ for the source node, and
3. There will be at least one value $< |G|$.

When condition (3) no longer occurs, the traversal is complete, and the locally smallest $|G|$ has been found. During this traversal, a set of nodes $n_{1,1}, n_{1,2} \dots n_{i,j} \in N$ is explored, each representing a different combination of C. Since each transform is likely to apply to several constituents – it could not reduce $|G|$ unless this were true, as shown earlier – it is also probable that it will be leveraged early on in the ZZ traversal through C and its usefulness will be visible by comparing values for $|G|$ during traversals with and without the inclusion of $t \in M$. Thus, a history H of $|G| \forall N$ where $M = \emptyset$ may be kept, and compared to $|G| \forall N$ where$M \neq \emptyset$ to identify where the continued search of C is unlikely to offer an improved $|G|$.

Naturally, this approach is not exhaustive; if a transform combination M applies to a subset $s \subset C$, and s does not provide a gain greater than $C \setminus s$, then s will not be selected during the traversal of C until after $C \setminus s$. Therefore, comparison of values for $|G|$ during the search would need to continue until $c \in s$ is reached, and search complexity would not be reduced greatly, if at all. Equally, if s provides greater gain than $C \setminus s$, it is possible to quickly detect that M allows for a smaller $|G|$ within N, and that traversal should continue as there is a possibility of finding a locally smaller grammar given the current M. With brief experimentation, it is possible to see that there is no single optimal value for $h = |H|$ for all inputs, but that a reasonable approximation is $h = \lceil 0.5|H(full)|\rceil$, and this may be reduced where appropriate.

To achieve this, for each input a ZZ traversal with $M = \emptyset$ is performed, beginning with $C = \emptyset$ and ending with $C \neq \emptyset$, and the values of $|G|$ for each selected node recorded in H, until $|H| = h$. Following this, a step is taken in transform space, adding $t \cup M$, followed by another traversal of the constituent space. When h steps have been taken, each a local minimisation $q = |G|$, the value $H[h]$ is queried; if $q >= H[h]$, the constituent space search is terminated, and q taken as the resulting $|G|$ for that node in transform space. Since $q >= H[end]$, it will not be selected as a local minimum of $|G|$, and traversal will continue through the transform space in search of a more effective combination of M and C.

If grammars are being constructed for a number of similar inputs, it is possible to begin with a high value for h and reduce it by a convex function such as gradient descent, to arrive at an approximate balance between minimal traversal steps of the constituent space and maximal use of transform in each M of the transform space. This may result in a more effective h, but adds greater overhead initially as $h \approx |H|$ until the descent reaches an optimal value.

Elimination of Non-Useful Transforms

During the search for substrings b in the input for which $b = f(T_{any}, a)$ can apply, a great many transforms $T_1, T_2 \ldots T_n$ are likely to be identified, where $n \gg 1$. If a transform T applies only to one instance of a substring b, it is not useful; as discussed previously, the cost of including and referencing T within G mandates that it apply to > 1 rule before it can offer any reduction in $|G|$. Such transforms may be immediately discarded from M before the search for optimal C and M combinations begins.

It is also possible to identify and discard transforms which apply to too few substrings to present any gain in the best case. Given a transform T with length $|T|$, and a set of associated substrings $u \subset S$ with quantity $|u|$ and whose lengths are $length(u_i)$, the maximum gain which is possible given T is $g = \sum_{i=1}^{|u|} length(u_i) - 2|u| - |T| + 1$. Therefore, the condition $g \geq 1$ must be satisfied to make it worthwhile to consider T. Where $g < 1$, T may simply be omitted from the transform space.

Limiting the number of Candidate Transforms

A further reduction to the transform search space may be made by retaining only the T with the highest probability of providing a reduction in $|G|$. The amount of gain possible in the best case can be used to roughly approximate this figure, and the transform space limited to an arbitrary number k of T. Once again, $g = \sum_{i=1}^{|u|} length(u_i) - 2|u| - |T| + 1$ can be calculated to produce a list of gains for all T, which in turn may be sorted in descending order, and only the top k entries retained. In this manner, the transform space can be constrained whilst approximately minimising the loss incurred from excluding any T which may otherwise have provided a reduction in $|G|$.

Equally, it is possible to seek transforms only for a specific set of lengths l_{min}, l_{max} for term a, such that $l_{min} \leq |a| \geq l_{max}$, where it is reasonable to believe that the range l contains useful transforms for $a \Rightarrow b$. A naïve search for a, b pairs here is reduced in complexity from $O((nl)^2) \forall l = |a|$ where n is the length of the string $a = S[i,j]$ with $0 \leq i,j \leq |S|$, to $O((nl)^2)$ where $l_{min} > i,j > l_{max}$ and $O(n^2 l)$ where $l_{min} \leq i,j \leq l_{max}$. Since every search for $b = f(T, a)$ could produce a potential match, and therefore require consideration during the ZZ traversals, computational complexity of grammar construction is also similarly reduced.

6.6 Examples

Appendix A presents two grammar encodings for Bach's Fugue No. 20 from WTC-II, the first including no rule modification, the second with modification enabled. A worked example of the construction of a transform-enabled grammar, for the chromatic pitch values of "*NLB073516_01*" from the Meertens Tune Collections (Meertens Instituut, 2018), may be seen in Appendix B.

6.7 Applications

A wide variety of applications exist for a grammar constructor which is able to generate smaller models, and which may segment not only by repetition but also by similarity, based upon modifications which may be customised to suit the nature of the input or task.

The method is broadly useful for general compression where a grammar-based compressor is desirable. Given the off-line nature of this class of algorithm, and since construction is computationally complex but recovery of the original input via model expansion is linear to the input's length, it is best suited for situations where compression is performed once but retrieval may occur frequently, and structure present within the input data is such that it may be leveraged to reduce a model's size. Since the addition of rule modification to a grammar's encoding increases substring similarity, the set of inputs to which grammar-based compression can apply in this scenario also increases, along with reducing the storage space required for each model.

Any task for which a straight-line grammar is useful may conceivably benefit from the application of grammars which allow rule modification. For example, the classification of digital musical scores into contextual groups such as composer, genre or family (van Kranenburg et al., 2016) may be possible with increased accuracy where, following the Minimum Description Length principle (Rissanen, 1978), a smaller model better represents the characteristics of the music and allows computation of a more accurate relative position within the contextual space. Equally, any grammar which is able to delineate segments with common domain-based relationships, for example musical devices such as inversion and retrograde, is better equipped to generate segmentations which are closer in structure to those an expert musicologist might produce (Bruhn, 1993). Where the structure of a grammar such as this is a better representation of a composer's intended structure, that grammar may prove more useful in retrieving themes and sections as a subtask of musical analysis (Meredith, David, 2013), particularly where those sections do not exactly repeat witin the piece.

Generally, any application to which a standard grammar containing exactly expanding rules may apply may benefit from the addition of rule modification, if the data being processed contains relevant patterns for which suitable transform can be devised, and discovery only of exactly-repeating segments is not the objective.

6.8 Summary

This chapter presented a novel approach to the construction of grammars containing production rules which may be modified during expansion, so they may generate outputs which are similar but not identical to their encoded term. This enables the creation of smaller grammars where substrings which approximately or "flexibly" match each other are present within the input string, by increasing substring similarity. The primary contributions of the presented method are the encoding of modifications which are applied to one or more production rules within the grammar itself, and the separation of the search space of constituent and modifier combinations into two individual domains. As demonstrated in Section 6.6, ZZ (Carrascosa et al., 2011) may be used to control the traversal of both search spaces, and a locally optimal constituent combination C selected for the best seen modifier combination M. The addition of a second search space increases complexity approximately by a power of 2, but heuristics such as early stopping, consideration only of modifications with a high probability of reducing the overall encoding length, and limiting the number of available modifiers, can help to reduce construction time. Nonetheless, further reducing computational complexity would be of benefit; Chapter 8 examines a technique by which this can be achieved.

The usefulness of the method, and how it functions given real-world data, may be best examined empirically. Chapter 7 explores its performance, and offers an evaluation of the properties of grammars which allow rule modification to occur, in particular the degree to which encoding length can be reduced.

Our empirical evaluation shows that we are able to find smaller models than the current best approximations to the Smallest Grammar Problem on standard benchmarks, and that the inferred rules capture much better the syntactic structure of natural language.

Payam Siyari & Matthias Gallé, 2017

7 Grammars allowing Rule Modification: Experiments & Results

In this chapter, the novel method of constructing grammars presented in Chapter 6 is empirically evaluated on three specific tasks: generation of compressed models from musical data; identification of musicologically-significant segments as defined by a MIREX *Discovery of Repeated Themes and Sections* task (Collins, 2016); and classification of folk music into "tune family" (van Kranenburg et al., 2016). The principal contribution of this chapter is a demonstration that the method is able to generate smaller encodings than ZZ (Carrascosa et al., 2011), despite the applied heuristics limiting the search space of rule modifier combinations which is explored during grammar construction, and proof that this reduction in model size is able to improve normalised compression distances to produce an increase in classification accuracy, supporting the hypothesis that smaller grammars are more appropriate models for this purpose. An exploration of the method shows a strong increase in construction complexity occurs as a result of the addition searches required to optimise the constituent combination C given each combination of rule modifiers M, and smaller models are generated for a minority of inputs. However, these models are able to improve task performance, and transforms chosen during the construction process are notably relevant to the musical context of the input they are applied to.

A novel aspect of this chapter is the evaluation of the performance of a grammar-based compressor allowing rule modification on a large corpus of musical works. Without a substantial population of results it is difficult, or even impossible, to define or understand the response of the method to a specific type of input. These experiments are designed to provide an indication of the suitability of grammars which feature rule modifications for use on musical data, and the level of performance which can be expected of the method, as a foundation upon which the improvements presented in Chapter 8, and future developments, can be based.

7.1 Introduction

Although grammars which allow for modifications to be applied to their rules during expansion appear interesting, their utility must be demonstrated through study and experimentation. Grammars have been shown to be useful in many applications, including generating compressed representations (Carrascosa et al., 2011), modelling musical structure (Giraud & Staworko, 2015), and seeking a hierarchy within DNA strings (Gallé, 2011). A key hypothesis in this study is that a representative segmentation of any intelligently generated or naturally occuring data sequence is very likely to contain related segments which do not exactly repeat, and are instead based upon transformed versions of themselves, potentially blending and overlapping with each other. Where this is true, the smallest model of the data will include such transformations, either directly within the encoding or by inheritance of existing model attributes. Although the type of grammar described in Chapter 6 does not allow rules to transition or overlap, segments in the data which do not make use of such features may be well represented by its transforming rules, providing the transformations considered correspond to those applied during creation of the sequence. As such, a partial validation of the hypothesis may be reached by demonstrating that such a grammar more accurately models the data, when compared to a straight-line grammar composed only of exactly-repeating segments.

In this chapter, several applications which are relevant to both exact and rule-modifying grammars are explored, and the performance of a standard grammar-construction method which is only capable of modelling exactly-matching rules is compared to the same method, but modified to incorporate the ability to model rules which may be transformed, resulting in the production of grammars which allow the modification of their rules during expansion, as described in Chapter 6. The purpose of these experiments is to evaluate the effectiveness of grammars which include rule modifiers in modelling real-world data, given a limited pool of potential transforms, with respect to grammars containing fixed rules only. This evaluation is intended as a contribution to current knowledge, demonstrating the ability of the novel and extensible technique presented in Chapter 6 to isolate appropriate, flexible patterns across a large collection of musical scores, and showing that this achievement often allows the production of smaller models which, in turn, can offer an improvement in performance on real-world tasks against which such grammars can be leveraged.

7.2 Experiment 1: Grammars as Compressed Models

Where the aim of compressing an input is to produce a compact encoding of that input, encoding length may be considered a reasonable measure of effectiveness. A compressor possesses characteristics arising from its algorithms, which often cause it to be more effective for a specific type of input data. For example, a dictionary-based compressor may be more effective on linguistic data where many repeating symbol groups are found, and less effective on a time series composed of real values. A limited ability to generalise may be seen where results for differing input types are compared; Siyari and Gallé's flexible matching approach did not generate smaller models than GA-MMAS given a corpus of DNA strings except in one instance, but was more effective for the majority of pieces in the Canterbury Corpus. The transforms discussed in Chapter 6 are designed with the intention of increasing equivalence within symbolic music, and so this experiment will concentrate on the scheme's effectiveness for musical data.

7.2.1 Purpose

The purpose of this experiment is to test the hypothesis that increasing substring similarity can enable more compact grammars to be generated, and that transforms of the type described in 6 are able to increase equivalence given strings of symbolic music. This idea relates to a hypothesis stated in Chapter 5 regarding the possibility of using grammars to identify musically significant structure in a score, insofar as a more compact model should be more capable of correctly identifying such structure, following the Minimum Description Length principle – this ability will be explored in the further experiments detailed in this chapter. Additionally, the ability to generate more compact models is an advantage where the general compression of musical data is intended, and grammars are chosen as the model type.

It is possible to explore the various attributes of a grammar which includes rule modifications by examining the models such grammar-based compressors generate. For example, there may be an imbalance in the effectiveness of different transform types, resulting in a tendency of the compressor to select only specific instances of each type, and this can be seen by looking at the distribution of transforms chosen over a large number of inputs. It is also possible that a grammar which does not include rule modifications may be more compact where there are few instances of the use of each transform type, limiting the ability of transforms to reduce encoding length. This experiment aims to examine the attributes of grammars which includes rule modifications, in addition to the encoding lengths which they produce.

7.2.2 Method

This experiment is conducted on pieces from the corpus of musical scores introduced in Section 3.6. For each piece, an input string S is generated from the chromatic pitch vales of each voice present within the input, with these sequences concatenated and separated by unique terminator symbols. S is then passed to two grammar-based compressors, the first exactly following the ZZ algorithm described in Chapter 4 and featuring no rule modifiers, the second also performing a traversal of an additional lattice of transforms, and able to generate grammars which include rule modification. The length of the input, $|S|$, is recorded for each piece, as is the output encoding size for the grammars generated by both compressors, and their construction times. The encodings themselves are also recorded, so that the types and attributes of any encoded transforms may be extracted and analysed.

As discussed in Chapter 5, the complexity of generating a compact grammar can lead to prohibitively high construction times, and this challenge is strongly magnified where an additional dimension of rule modifications must also be traversed. In order to allow as many inputs as possible to be evaluated in the time available, pieces are processed in order of increasing approximate complexity, defined in Chapter 3 as the run-times required to produce standard grammars for each using ZZ. Given that there is a direct correlation between this definition of complexity and piece length, any bias which might be introduced by failing to process the largest pieces should be minimal, providing a reasonably large sample is processed, and there is no indication from the results of a trend towards a decreased effectiveness for grammars which allow rule modification as input size increases. In theory, such bias should not exist; the addition of useful transforms should increase the number of substrings which can provide compression, creating an effect similar to providing ZZ with an input which contains greater exact repetition: the models produced using grammars which allow rule modification should be smaller than their ZZ counterparts, and the effectiveness of the transforms chosen may be observed by their inclusion in such models.

7.2.3 Results

Of the 7928 pieces in the corpus, it was possible to process 5521 in the time available, thus covering a large proportion of the available data.Nonetheless, the omission of 2407 pieces altered the distribution of attributes, primarily resulting in the inclusion of approximately only half of scores from the corpus belonging to the classical genre, and allowing folk pieces to become the dominant group for the ex-

Composer	Proportion
Unknown	*42%*
Bach, Johann Sebastian	*15%*
da Palestrina, Giovanni Pierluigi	*9%*
Others (<1% each)	3%
Haydn, Joseph	<1%
Corelli, Arcangelo	<1%
Mozart, Wolfgang Amadeus	<1%
Beethoven, Ludwig van	<1%
Vivaldi, Antonio	<1%

Table 7.1: Distribution of the 5521 pieces processed during this experiment by composer, relative to the entire corpus; three primary composers continue to form the majority of the inputs, although the number of pieces by all named composers is strongly reduced overall.

periment. Tables 7.1 – 7.3 show the altered distributions as proportions of pieces relative to the entire corpus (as shown originally in Tables 3.2 – 3.4). Aside from the strong reduction overall in classical pieces, some notable changes are the omission of all but four pieces by Haydn, the loss of approximately 30% and 52% of pieces sourced from *Music21* and *KernScores* respectively, and the retention of over 87% of pieces from the medieval period.

For the majority of inputs, no benefit arose from the addition of rule modifiers given the chosen constraints, resulting in an equal model size being produced by both methods for 75.46% of all inputs (4066 of 5521 grammars). No model including rule modifiers had an encoding size in excess of its fixed-rule counterpart. Figure 7.1 shows that there is a general trend to smaller model size ratios for grammars which allow rule modification, compared to those which do not. However, it is important to note that each curve's values are sorted independently, and any single smaller model size results in a continuous offset on the figure. Representation of the figure in this manner is necessary due to the high piecewise variation in encoding lengths observed during the experiment, resulting in a chaotic sequence of values in one array used to plot the figure if it is sorted according to the indices of the other, but independently sorting the arrays in this manner allows for observation of an over-arching trend across the population; generally, for each model which included rule modifications, the corresponding model without rule modifications was larger in encoding length. Despite the large number of models which did not benefit from rule modification, a reduction in encoding size for 24.54% of all models remains a significant amount, and it is possible with different constraints or additional, more relevant transforms available that a greater proportion of all inputs may generate smaller models.

Figure 7.2 charts grammar construction times for both methods, with values sorted independently

Period	Proportion
Undefined	*43%*
Baroque	*15%*
Renaissance	*9%*
Medieval	1%
Classical	<1%
Romantic	<1%
Georgian	<1%
Tudor	<1%

Table 7.2: Distribution of the 5521 pieces processed during this experiment by period, relative to the entire corpus; the three largest continue to be Baroque, Renaissance, and "undefined" (where creation dates were unavailable), but the number of pieces is almost halved for the *Baroque* and *Renaissance* periods.

Genre	Proportion
Folk	*41%*
Classical	*27%*
Undefined	1%
Jazz	<1%

Table 7.3: Distribution of the 5521 pieces processed during this experiment by genre, relative to the entire corpus; the majority of scores are now folk pieces, although just under half of all classical pieces from the corpus remain.

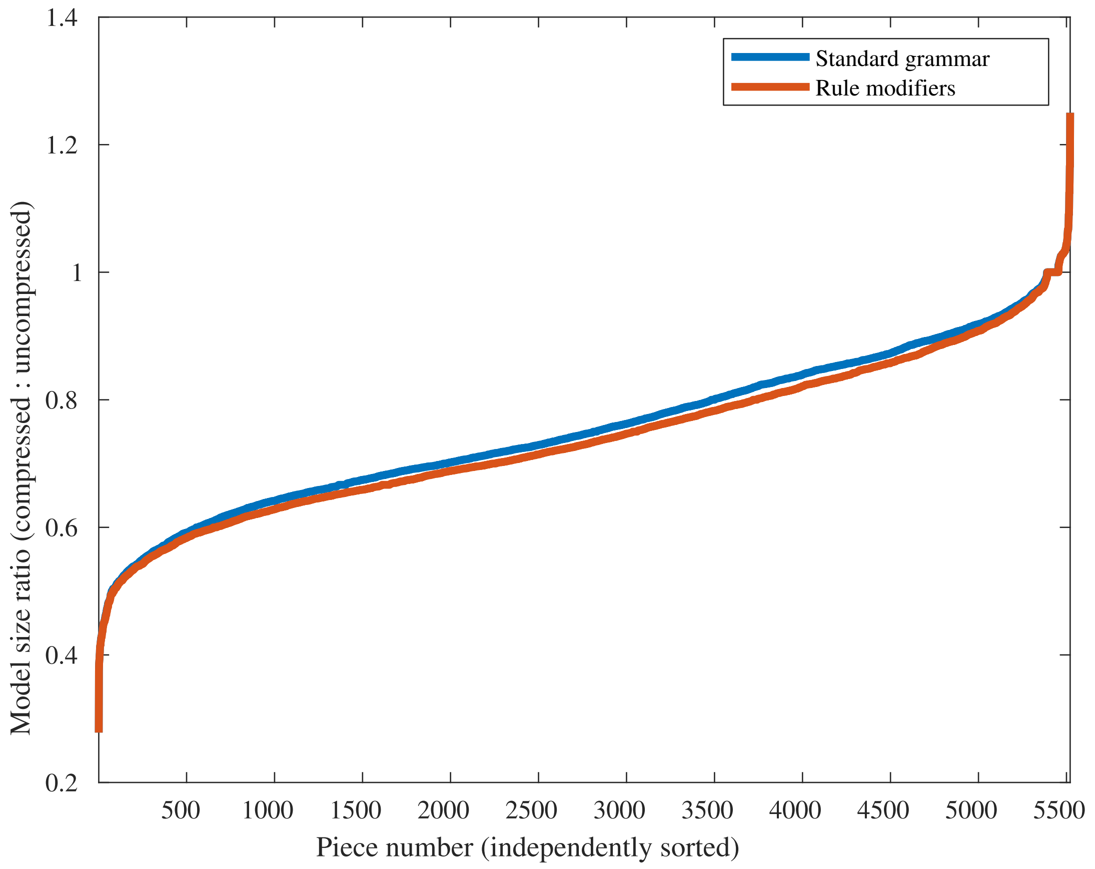


Figure 7.1: A comparison of compression ratios for the target pieces, for both standard grammars and those which allow rule modification. Ratios are sorted independently for each method, and presented as a response curve which is relative to the size of the input which the model represents.

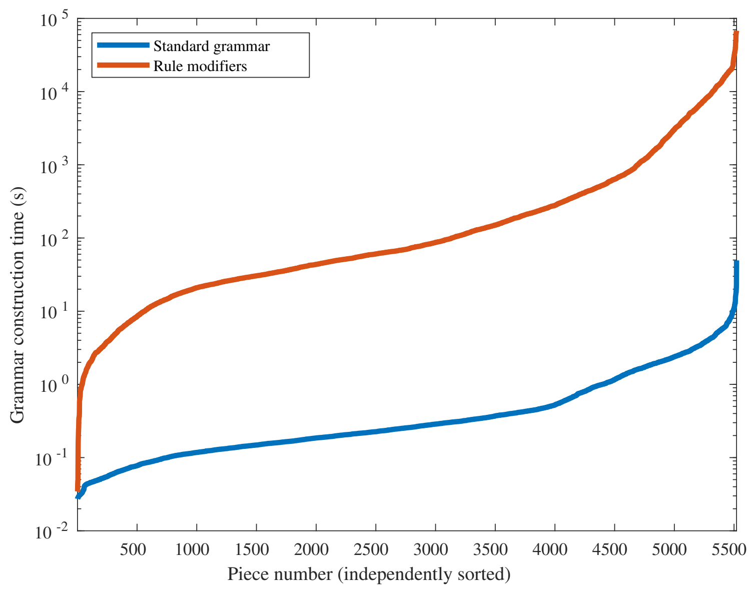


Figure 7.2: A comparison of grammar construction times in seconds for the target pieces, for both standard grammars and those which allow rule modification. Times are sorted independently for each method, and presented as a response curve over all pieces tested.

and plotted against a logarithmic scale to provide a comparison of run times in practice. For both traces, data input length undergoes an overall increase, with the shortest pieces represented by leftmost data points, and the longest by those on the right. As the figure demonstrates, there is an exponential increase in construction time when grammars are built using rule modifiers, and this increase is proportional to the construction time for standard grammars.

Difference in generated model sizes between the two methods is generally consistent; standard deviation is within 8.78 symbols, with a maximum difference of 103 symbols across all pieces tested. For models including rule modifiers which offered an improvement over fixed-rule models, standard deviation is within 12.0, with an average of less than 15.0. The most compressed model was less than 0.279 times the size of the input, and this was not improved by the addition of rule modifiers – the input was amongst the largest in size at 266 symbols (the maximum tested had a length of 879 symbols). The most compressed model which benefitted from the addition of rule modifiers was less than 0.653 times the size of the input, which was of length 407 – this piece was amongst the top 7% of largest inputs tested. From the set of all models whose encoding length was reduced by the addition of rule modifiers, the average increase in compression was $> 5.47\%$, with a minimum of $> 0.14\%$, a maximum of $> 22.6\%$, and a standard deviation of $> 3.18\%$. Figure 7.3 shows the range of improvements

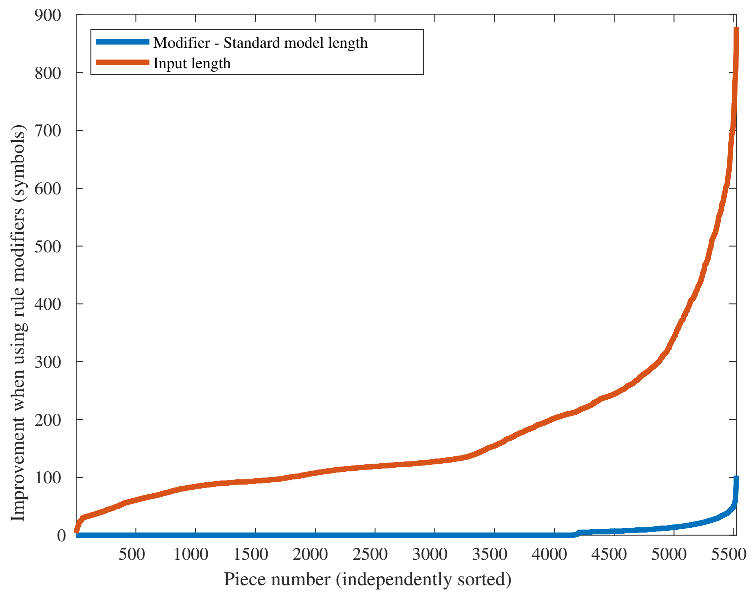


Figure 7.3: Reduction in model sizes observed when choosing grammars which allow rule modification over those which do not, in comparison with the lengths of input tested. All y-axis values are in symbols.

rule modifiers offer over standard grammars in comparison with the range of input sizes processed.

Where the addition of rule modifiers was able to offer a reduction in encoding size, compression ratios ranged between 0.42 – 0.963 as may be seen in Figure 7.4. For models which could not be improved, ratios ranged between 0.2782 – 1.25, with all models whose ratios are greater than 1.0 being composed of *S*\$, where *S* is the input string and \$ is the encoding terminator symbol. Note that ratios converge at 0.42, suggesting the most compressive pieces in the corpus may be composed of substrings which strongly and exactly repeat.

Figure 7.5 shows the distribution of the transforms chosen, from all 802 grammars whose encodings included rule modifications. From these, 63 unique transforms were used, with 32 individually appearing in only one model. For clarity, the figure only shows the distribution of transforms which appeared in at least 5 grammars; the remainder included transforms which deleted rule elements, substituted them, translated them by a specific amount, reflected them around the axis of a given value, or – in 2 instances only – combined one of these operations with an initial reversal of the sequence generated by the rule.

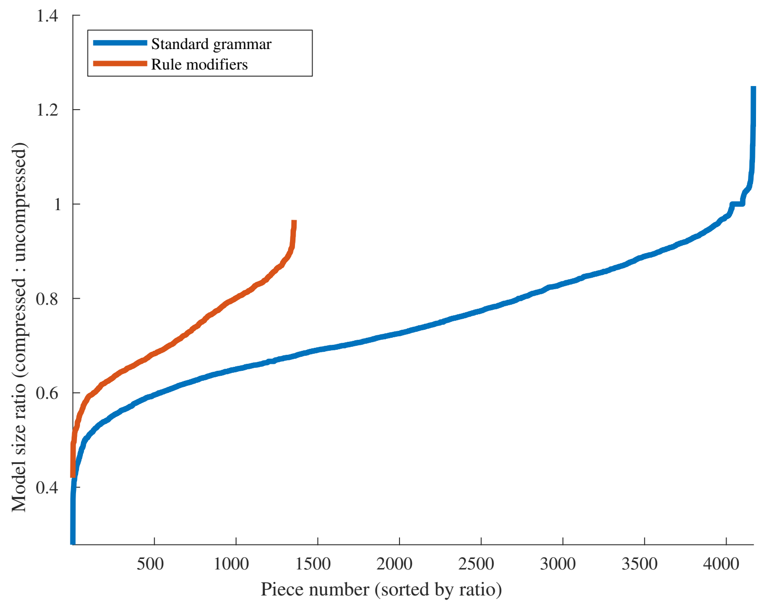


Figure 7.4: Compression ratios for the pieces for which the addition of rule modifiers offered smaller models. Values are sorted by ratio in ascending order, and presented as a response curve which is relative to the size of the input which the model represents.

7.2.4 Analysis

Given that the majority of inputs did not benefit from the addition of rule modifiers to their models, it may reasonably suggested that either no further reduction beyond a standard grammar is possible for these inputs, or that the constraints applied during modifier-enabled grammar construction were too narrow. From examination of the grammar-based segmentations shown in Chapter 5, Section 5.6, it is clear even with very little knowledge of music analysis that further equivalence exists between segments – for example, the *Subject* sections of voice 1 in Bach's WTC I Fugue No. 2, figure 5.10 – and that these are likely to produce a shorter explanation of the piece than if exact equivalence is used alone. This suggests it is probable that the addition of further types of modification, perhaps based specifically on analytical techniques from the musical domain, could allow more compact grammars to be generated and thus increase the proportion of pieces in the corpus which would benefit from rule modification. Equally, inputs with a different context may benefit from other, specifically-targeted transforms. However, the experiment itself does not provide anything beyond supposition to this theory. It does, however, prove that rule modification is both appropriate and useful to a large minority of musical inputs, even when heavily constrained and consisting only of quite simplistic transforms.

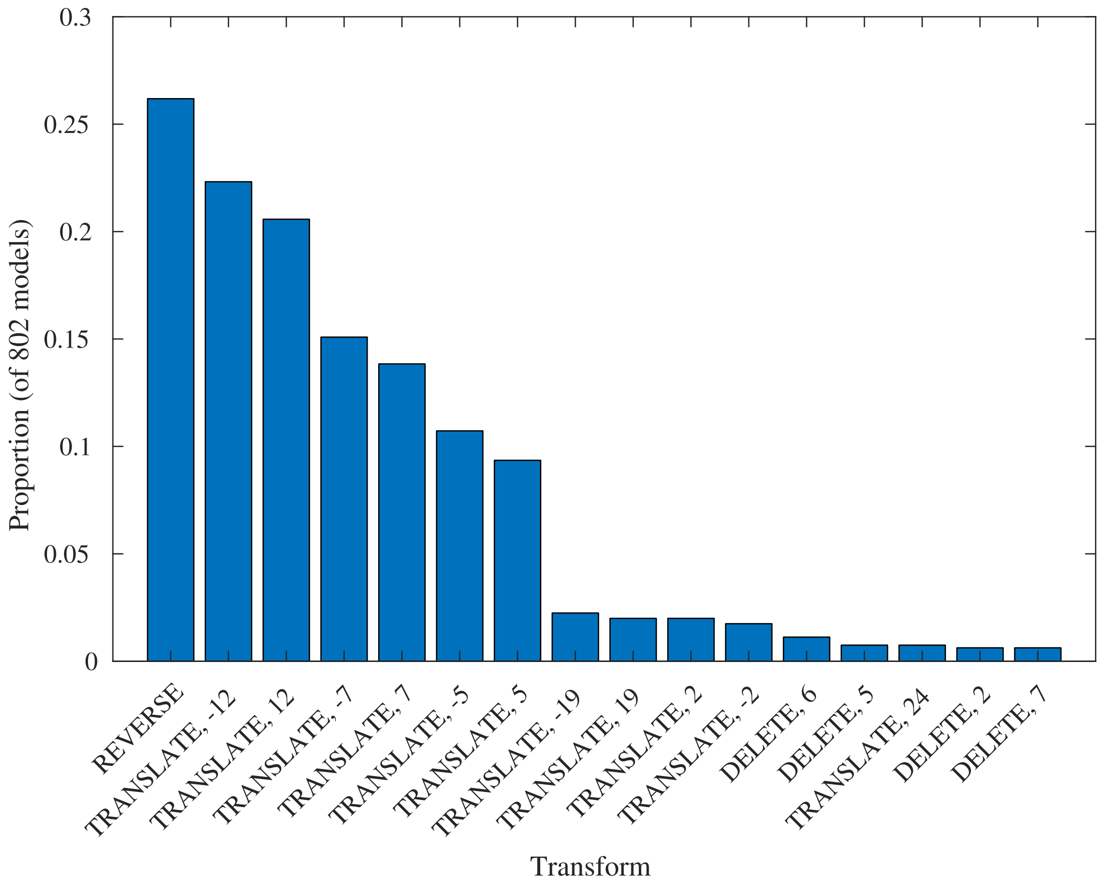


Figure 7.5: Distribution of transforms chosen during generation of grammars which included rule modification, with number of occurrences shown as a proportion of the count of these grammars. Only transforms which occurred in 5 or more models are included in the figure for clarity.

Construction times for grammars which include rule modification are prohibitive – the additional dimension of modifiers which must be traversed causes an exponential increase in computational complexity, proportional to the size of the set of transforms which are to be tested for potential inclusion. This increase is especially noticable for larger inputs, but does affect *all* input lengths. The greater the number of possible transforms which must be considered, the greater the increase in construction complexity, but equally the greater the potential for leveraging a transform to reduce the size of the model constructed. This clearly demonstrates, for this version of the algorithm, that grammars which allow rule modification are appropriate for offline applications where generation of a smaller model is desired, for compression or analytical purposes, but fast construction times are not required. Whether it is possible to limit the additional complexity caused by the introduction of transforms through discovery of an alternative construction algorithm, or whether this complexity is inherent as one moves closer to the theoretical minimum possible encoding and with diminishing returns, is, unfortunately, a question which is beyond the scope of this study.

In compression terms, the benefit of allowing rule modification is not great at a reduction of over 5% average on the tested inputs. However, it is not minimal, and at an average encoding side over 8 symbols less than the equivalent standard grammar encoding, it is reasonable to suggest that there may be a connection between the context of the segments formed by production rules and the number of symbols saved. For example, it is common to find a group of quavers fully occupying a bar with a 4/4 time signature, and a reduction of 8 symbols in a musical encoding could signify anything from the addition of single notes to existing rules, to identification of an entirely new multi-note segment, for which one or more transforms have enabled equivalence. Given that the transform itself incurs a cost within the grammar's encoding, more than 8 notes must be leveraged to gain such a reduction, suggesting the segments involved in the reduction are likely to have some contextual significance in connection with the transform types involved (and are not merely random, minimal gains made possible by the transform in general). In the best case, a reduction in encoding size of over 22% was possible, for a model of length 105 symbols, with an original input length of 181 symbols. Without the use of rule modification, a grammar of length 146 was generated, showing a total saving of 41 symbols was possible for this particular example. These observations show grammars which include transforms are certainly useful in generating models of reduced size, and these models *may* contain rules with a greater contextual relevance, although this experiment alone does not itself prove that any such relevance exists.

Transform Type

It is of particular interest that the context of the transform types most often chosen for inclusion in the grammars which include rule modification is, arguably, strongly musical; if this characteristic were not present, it might highlight a failure of the method to discover structure known to be present within the data being processed, which is likely of great significance to a concise explanation of its configuration. As shown in Figure 7.5, the most common transforms occurred in approximately 10% - 25% of all such grammars, and were often complementary, for example representing translations of both ± 12 semitones. The most popular transform, a full reversal of the production rule's sequence, may be considered a generalised operation. Within a musical score, however, it may represent the latter half of a sequence which ascends and then descends, and may equally apply to a motif which passes into a harmonic passage, and is later used in the opposite temporal order to close it.It is also important to note that a reversal carries a cost of only 1 symbol (plus terminator) for inclusion as a modifier, less than all other chosen transforms, and this causes a bias towards its selection during grammar construction. A more concrete statement may be made regarding the following six transforms: three individual operations are represented, and the group contains inverted versions of each. These transforms take the sequence of values a production rule generates, and numerically translates them, shifting their value by the amounts given, in this case ± 12, ± 7, and ± 5. From a musical perspective, these values are highly significant, given the use of chromatic pitch values in the input data. The ± 12 represents an increase or decrease in pitch of exactly one octave. A shift of $+7$ or -5 represents a raise or drop in pitch from the tonic note by a perfect fifth – an extremely common melodic movement, as may be seen in the perfect cadence. Similarly, a change of -7 or $+5$ results in a step of a fourth from the tonic note. Given that steps of fourths and fifths are very often found within classical music, of which the inputs in this experiment are primarily formed, the frequent use of such transforms to generate more compact encodings is unsurprising, and points towards correct identification of several underlying pitch relationships by the compressing program.

Examination of the figure's remaining transforms yields similar relationships – a shift of ± 19 is likely to represent compound translations, of either an increase of an octave plus a perfect fifth, or a decrease of an octave plus a fourth, from a tonic pitch. Limited use was also made of ± 2 semitones; if it is assumed that these transforms are applied to the tonic, the resulting pitch would be either a second – a common interval in an ascending harmonic progression – or a seventh, which may be found in some descending patterns and phrases. A translation of $+24$ is also present; this may have allowed production rules which represent patterns primarily found on a bass clef to also produce identical patterns on a treble clef. The remainder of the transforms relate to removal of pattern elements, for individual indices of the second, fifth, sixth and seventh notes in a sequence. Without close examination of the patterns which these transforms are applied to it is impossible to provide a definite description of the context and purpose of the operations performed, but it is certainly reasonable to assume that a less complex or cut-down version of a pattern may appear elsewhere within a musical score. For example, a basic progression of *C, E, G* may later appear in a shorter form as *C, G*, where the note at the second index is removed, and such variation may be seen in a great many works. It is perhaps most likely that the positions 2, 5, 6 and 7 refer commonly to the last note within a pattern, allowing other instances whose final note is either changed or absent to remain a match.

Overall, the experiment shows that a small group of transforms were most commonly chosen, and given their nature it is highly probable that their relevance in the context of the inputs' domain is responsible for their usefulness in generating compact models. The fact that their presence was able to reduce encoding length supports the hypothesis that domain-based transforms are relevant to the production of smaller grammars, and that the construction program was able to select them specifically from the wide range of available modifications demonstrates the ability of a compressor to discriminate between useful and non-useful operations. Certainly, the construction of smaller grammars is on occasion more successful where such transforms are considered for inclusion within their encoding.

7.3 Experiment 2: MIREX 2016 Discovery of Repeated Themes & Sections task

As previously discussed, the MIREX *Discovery of Repeated Themes and Sections* task explores the effectiveness of an algorithm to identify patterns selected by experts as musicologically significant from within a small collection of musical scores. The Johannes Kepler University Patterns Test Database (Johannes Kepler University, 2013) contains the necessary data, segmentations and evaluation code. In this experiment, grammars which include rule modification are applied to the task, and their performance compared to other existing methods.

7.3.1 Purpose

The primary purpose of this experiment is to test the hypothesis that a grammar construction algorithm which is able to apply compositionally-relevant transforms between segments is more likely to correctly identify patterns in a musical score, in a similar manner to an expert musicologist. Such patterns may be said to be structurally significant to the composition, and musically important to the piece's context. In the absence of a large corpus of musical analytical data against which performance for the various analytical techniques may be tested, the MIREX 2016 *Discovery of Repeated Themes and Sections* task is chosen as an approximation, albeit with a very small sample of the population of musical works.

7.3.2 Method

Following the previous experiments detailed in Section 5.6.3, grammars which may contain rule modifiers are constructed from each of the pieces in the Johannes Kepler University Patterns Test Database (Johannes Kepler University, 2013). Due to the high complexity of constructing grammars allowing rule modification, pieces are processed in order of approximate complexity, as discussed in Chapter 3, Figure 3.1, to enable the largest possible number of inputs to be processed. Each sub-rule of S is expanded using all its associated transforms $T_n \in \mathcal{M}$, to produce a set of all used variations, and these fully expanded instances are passed to the MIREX 2016 code designed to evaluate algorithm performance. The polyphonic version of the symbolic task has been selected for investigation, and the focus is again on the metrics *establishment* and *occurrence*. As defined by the provided evaluation procedure, matches with a score threshold ≥ 0.75 are selected as positive identifications. F-measures are calculated from each metric, and these are compared both to the official results for 2016, and to the results obtained using ZZ in Chapter 5, to discover whether the addition of rule modifications enables any improvement in performance.

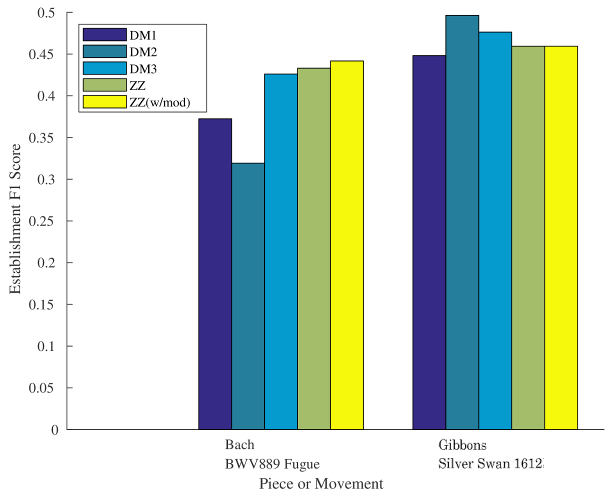
7.3.3 Results

In the available time, only the two least complex pieces could be processed: Bach's Fugue No. 20 from *Das Wohltemperierte Clavier* Book II, and Gibbons' *The Silver Swan*. The results are shown in Figures 7.6a and 7.6b. The addition of rule modification did not allow production of a smaller model for Gibbons' *The Silver Swan*, and so no improvement in the F1 scores occurred for this piece. However, grammar size decreased from 583 to 411 for Bach's Fugue No. 20 with the addition of rule modification. A human-readable version of these encodings may be seen in Appendix A.

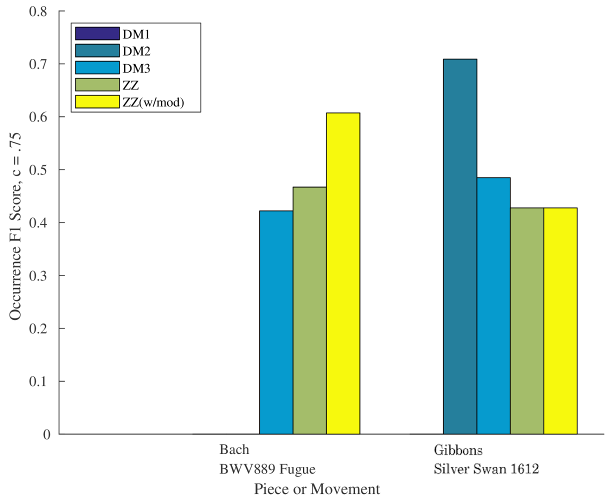
The number of rules decreased from 40 to 36, and 9 transforms were selected: a reversal, deletion of an 8^{th} element, translations of -24, -5, 7, 10, 12 and 17 semitones, and a translation of -7 semitones followed by deletion of the 1^{st} element. F1 scores for this piece were increased where rule modification was used, and in particular a strong increase was observed for pattern *occurrence*, from 0.47 to 0.61.

7.3.4 Analysis

Because a less compact model could not be found for Gibbons' *The Silver Swan*, the lack of improvement in F1 scores is, of course, to be expected. It is possible that selection of a different set of modifier constraints might allow a greater or different transform search space to be explored, and could result in the discovery of a smaller model, but whether this could produce an improvement in performance for



(a) *Establishment F1 score*



(b) *Occurrence F1 score*

Figure 7.6: MIREX 2016 Discovery of Repeated Themes & Sections symPoly task, Establishment & Occurrence F1 scores for 2016 algorithms DM1, DM2 & DM3 with additional results for ZZ, and ZZ with the addition of rule modification. An occurrence F1 score of 0 results from failure of an algorithm to identify at least 75% of the total instances of any pattern - no bar is plotted for these cases.

the piece is left for future work. The increase in F1 score for Bach's Fugue No. 20 from *Das Wohltemperierte Clavier* Book II is certainly linked to the difference in production rules which occur within the more flexible grammar, and it is reasonable to suggest the improvement is caused by the ability of the grammar to create a segmentation which is more musically significant to the task.

Where a transform is applied to a production rule, an expansion of that rule which differs from the rule's encoding is output. As such, any rule which is modified generates a different segment of the input score, thus associating it with a greater overall coverage of the input sequence. The *Discovery of Repeated Themes & Sections* evaluation code scores an algorithm more highly where it returns a greater number of segments which have been identified as significant by expert musicologists, and so it is logical that an algorithm which selects a greater number of segments may score better in recall terms. However, precision is also high for this piece, showing that the segmentation is relatively accurate in discriminating between segments which are contextually significant, and those which are not. For the *establishment* metric, precision and recall are 0.29 & 0.82 for ZZ respectively, and 0.3 & 0.84 for ZZ allowing rule modification. For *occurrence*, precision and recall are 0.89 & 0.32 for ZZ, and 0.9 & 0.46 for ZZ allowing rule modification. These figures highlight the more flexible grammar's increased ability to identify occurrences of the target pattern in the piece, due to the larger number of segments each rule represents. Indeed, F1 score for *occurrence* is higher for the grammar which includes transforms than any other method, including those submitted to MIREX 2016. This result clearly supports the hypothesis that greater rule flexibility can create a more structurally-significant segmentation, although the single result alone is insufficient to allow a generalisation of far greater scope to be suggested. The transforms which were chosen by the algorithm during grammar construction have enabled the generation of a far smaller grammar, potentially showing they are highly to the musical domain of the input sequence.

7.4 Experiment 3: Classification of the Meertens Tune Collections by "tune family"

This experiment continues the comparison of results from tests conducted in Chapter 5 by applying grammars which include rule modification to the task of classifying the Meertens Tune Collections *Annotated Corpus v2.0.1* (van Kranenburg et al., 2016) by "tune family". As a widely-investigated task, a great deal of context exists against which to analyse results, and it is of course useful to place grammars which include rule modification into this context.

7.4.1 Purpose

The primary purpose of this experiment is to test the hypothesis that a grammar which is able to include musically-relevant transforms is better able to model differences and similarities between pairs of musical inputs. As an approximation of similarity between two musical scores, normalised compression distance may be used to compute a pairwise measure which may then be used in the classification of a collection of inputs. Since the compressor ideally selects the combination of patterns which produce the smallest possible model which describes two scores, it is reasonable to hypothesise that extending the compressor's ability to choose segments in a musically-related fashion may allow smaller models to be generated, which in turn may be used to measure similarity more effectively.

7.4.2 Method

The method for this experiment closely follows that given in Section 5.5. Grammars which are able to make use of rule modifications are constructed by the compressor C for each of the 360 scores in the Meertens Tune Collections *Annotated Corpus v2.0.1* (van Kranenburg et al., 2016) in a pairwise fashion, and three models are constructed for each: $C(x)$, $C(y)$, and $C(xy)$, where xy represents a concatenation of x and y separated by a unique symbol. A Normalised Compression Distance (Li et al., 2004) is then computed for each pair of scores, as defined in Chapter 5 in Equation 5.1. Due to the complexity involved in the construction of a grammar which includes rule modifications using the method described in Chapter 6, the only representation given to C is chromatic pitch values. Due to this, results are compared only to the performance of the ZZ-based method *without* rule modification enabled, and given the same representation, and others tested in Section 5.5 are not considered. Comparison of performance based on other representations is left for future work.

Classification into "tune-family" as defined in the corpus is performed using the 1-Nearest-Neighbour algorithm (Cover & Hart, 1967), producing distance-based clustering, and leave-one-out cross-validation (Kohavi, 1995) is used to evaluate accuracy against the ground truth included in the collection; this process closely follows that of Meredith (2014). Success rate r is calculated simply as per Equation 5.2, where c is the number of correct classifications and t is the total number of pieces tested. The results obtained are compared directly to those discussed in Section 5.5.3, but only for a chromatic pitch input representation.

7.4.3 Results

The accuracies with which ZZ-derived NCDs could be used to classify pieces from the MTC-ANN v2.0.1 are shown in Table 7.4, both with and without rule modification enabled during grammar con-

Table 7.4: Rate of successful classification of pieces from the MTC-ANN v2.0.1 for ZZ, with and without rule modification enabled.

Rule Modification	Standard
<u>0.875</u>	0.8583

struction. Where grammars which include modifiers were used to construct the models, classification accuracy improved by of 1.7%.

Of the 360 pieces in the collection, sizes of models constructed from an individual piece varied for only 4 models, where the addition of rule modifiers reduced encoding length by 6, 6, 4 and 2 symbols respectively. This alteration affected just over 1.1% of all inputs. Of the 360^2 pairwise combinations, smaller models were generated for 4776 inputs, representing 3.69% of all pairs. For the latter group, the average reduction in encoding length was approximately 5.8 symbols, with a standard deviation of 2.48. In the best case, model size was reduced by 24 symbols.

Of the pieces which could not be further compressed individually by the addition of rule modifiers, 3.3% of all pairwise models exhibited reduced encoding lengths when compared to their counterparts without rule modifiers. On average, encoding size was reduced by 0.193 symbols, with a standard deviation of 0.4. The best case reduction of 24 symbols fell within this group.

When distances were computed from the model sizes obtained both for single and concatenated pairs of inputs, a greater number of differences were observed within the distance matrix: 5178 elements were less within the matrix calculated from models which included rule modification, and 2044 were greater, representing just under 4% and 1.58% of all elements respectively. Of all possible distances, 5.57% were altered through the presence of more flexible grammars. The average distance within both matrices was 0.476245 when calculated from standard models, and 0.4753 when calculated from models which included rule modification, with standard deviations of 0.0285 and 0.0294. The average reduction in distance within the latter matrix was 9.12×10^{-4}, with a standard deviation of 0.008329, representing a change of 0.0167% from the original distances.

When used to classify the collection, predicted class differed for 26 pieces, representing $> 7.2\%$ of the corpus. Of these, 10 were newly assigned to an incorrect class, and 16 were newly assigned to their correct class, providing an overall increase in the number of correct classifications by 6 pieces. Of the newly correct classifications, 1 was associated with an input whose individual encoding was reduced through the addition of rule modifiers, and all 16 were related to reduced distances within the matrix of NCDs.

7.4.4 Analysis

Although the degree of classification accuracy is similar, the addition of rule modifications has altered the resulting grammars sufficiently to provide a minor improvement in classification ability of +0.0167. Given that 360^2 measurements exist within the matrix of Normalised Compression Distances used during classification, and that 5.57% of these of these offered potentially more relevant distances due to the presence of transforms within their related encodings, it might be reasonable to anticipate a roughly proportionate improvement in classification accuracy. In the best case, this may lead to an ability to correctly classify a further 5.57% of all pieces, resulting in $\lfloor 360 \cdot 0.0557 \rfloor = 20$ additional correct classifications, resulting in an accuracy of $\frac{329}{360} = 0.9139$. In the worst case, changed distances may be associated with already correctly-classified pieces, resulting in no improvement. The actual observed accuracy of 0.875 is less than the best-case expectation, but greater than the worst case. Thus, it may be stated with certainty that altered distances are associated with some pieces which were previously incorrectly classified, and the change, although minor at an average of -0.1672%, is sufficient to produce an improvement on the task. These results suggest that even a minor difference in model sizes used when calculating NCDs may be significant to classification accuracy, and thus the more compact grammars produced when rule modification is permitted are relevant to such a task.

It is important to note that the change in distances caused 10 pieces which were previously correctly classified to become incorrectly identified. Of the output classes which were affected by the changed distances, over 38% exhibited degraded performance, whilst over 61% were improved, representing 2.78% and 4.44% of all inputs respectively. This single experiment cannot provide sufficient proof that use of more compact grammars is likely to offer improved classification in the majority of cases, although in this instance accuracy was shown to have increased. Standard deviation is notably high for the difference in distances between the two methods, potentially signifying a relationship between this wide variation and the number of degraded – as well as improved – class selections. However, in the majority of cases changes in distance resulted in an increase in performance, and this is likely connected to the measured average reduction in distance.

A significant outcome of this experiment is that the use of grammars which include rule modification can allow calculation of smaller NCDs, even when they cannot offer a reduced encoding length for most individual inputs alone. The pairwise concatenation of inputs presents a greater opportunity for equivalence given the availability of transforms, and greater equivalence in turn may lead to an increased ability to discriminate between similar and dissimilar inputs.

7.5 Summary

This chapter presented a novel comparison between the performance of two grammar-based compressors using ZZ optimisation – one which traverses an additional space of transforms in order to produce grammars which include rule modifications, and one which does not – on three real-world tasks: compression of musical inputs, identification of musicologically-significant segments of musical scores, and classification of folk songs by "tune-family" using normalised compression distance. The presented experiments explored the effect of augmenting a particular grammar-based compressor with the ability to select custom rule transformations. Overall, experimental results showed the inclusion of rule modifers was able to reduce grammar encoding lengths for a significant minority of inputs, and an improvement in classification accuracy was observed when the smaller models were used to compute pairwise distances, supporting the hypothesis that more compact models more accurately represent the structure and properties of their input data.

When applied to the task of compressing pieces from a large corpus of digital scores, the inclusion of rule modification enabled more compact models to be built for $< 25\%$ of all inputs, which were on average 5% smaller than their standard counterparts. Despite the imposition of a limit on the number of transforms which may be considered for inclusion in each grammar, in an attempt to reduce time complexity, a strong increase in construction times was noted, and the following chapter introduces a method by which ZZ-based grammar optimisation may be accelerated to help address this challenge.

Where the production rules belonging to grammars constructed from two of the five pieces from the Johannes Kepler University *Patterns Development Database* (Johannes Kepler University, 2013) were taken as candidate note sequences, and compared against those identified by expert musicologists, a definite improvement in identification ability as evaluated by the task's metrics was observed over the use of production rules from standard grammars. However, the encoding length of one piece only was reduced by the addition of transforms to its grammar. Improvement on the task occurred because the inclusion of transforms allowed production rules to represent a greater number of musically-relevant segments. The inability of the new method to generate a smaller encoding for only one input was not unexpected: only a minority of pieces from the corpus of musical scores benefitted from the inclusion of rule modifiers, and that particular score from the *JKU-PDD* did not fall within the group.

Grammars allowing rule modification were constructed for each piece in the Meertens Tune Collections *Annotated Corpus v2.0.1* (van Kranenburg et al., 2016), and from pairwise concatenations of each piece. The resulting model sizes were then used to compute a pairwise matrix of normalised compression distances, and this matrix used to classify each piece by "tune-family" using the 1-Nearest-Neighbour algorithm (Cover & Hart, 1967). The process was repeated using standard grammars, and the classification accuracy each method achieved compared. An improvement was observed where models were constructed which included rule modification, and classification accuracy increased from 0.858 to 0.875. Since the presence of transforms produced the improvement, the results do not reject the hypothesis under test. However, the sample size is small, and the outcome cannot conclusively prove its validity. Further testing on a variety of larger problems is desirable, as future work.

These results generally demonstrate that the addition of rule modification to a grammar construction algorithm can, in principle, allow the production of smaller models, which in turn can be used to produce interesting, improved results on a variety of tasks. The technique provides an alternative to existing flexible-matching approaches, such as the encoding of prefix and suffix variations (Siyari & Gallé, 2017) in a non-recursive grammar, and allows the manner in which rules may be altered to vary depending on the set of transforms defined at the time of construction. The results show the transforms chosen for these experiments are useful in the compression of digital score data. Construction complexity is, however, prohibitive; the following chapter presents a novel method of lattice traversal which seeks to improve this shortcoming.

First, we improve the complexity of existing algorithms by using the concept of maximal repeats when choosing which substrings will be the constituents of the grammar.

Rafael Carrascosa et al., 2012

8 An improved method of Lattice Traversal for Grammar Construction

8.1 Introduction

A key consideration for any compression scheme designed to operate on real-world data is its time complexity. If execution time were of no importance, an ideal compressor – one which always finds the most highly-compressed model from the universe of all possible models – could easily be designed, even if an exhaustive exploration of its search space were required. Conversely, an ideal compressor in temporal terms is equally simple: whilst any alteration may be made which reduces the size of a given model, the best change which is immediately available can be selected from the space of all possible changes, even where further exploration is likely to result in greater compression, and this process continued until no obvious improvement exists. In this example, the resulting model can be said to represent the encoding which may be generated most quickly. Unfortunately, neither scheme is efficient; an exhaustive approach may prove impossible or simply impractical to compute, and an entirely greedy approach is likely to produce an unnecessarily large encoding of little practical use or compression.

ZZ (Carrascosa et al., 2011) provides a logical path to achieving a balance between speed and efficiency. The space of all possible changes is constrained to the addition of a single constituent to a current encoding, and the result of each possible addition is assessed by generating the encoding itself, ensuring the actual outcome of each option is known prior to the choice being made. The algorithm is greedy in the sense that a constituent which may prove less optimal overall, for example by preventing the addition of two others which offer greater combined compression, will still be chosen if it offers an immediately greater gain. However, it retains the possibility of later rejecting the sub-optimal constituent, if its removal results in a more compact encoding at that later time. Unfortunately, for the rejection to occur, the combination of constituents which it blocks must be already present within the encoding, so that these become fully active when each possible rejection is tested. Since such blocked constituents do not present sufficient gain in terms of model compactness to warrant their earlier inclusion, the circumstance cannot arise, and only a locally optimal encoding may be discovered by ZZ. Regardless of this, the algorithm remains a benchmark in straight-line grammar construction, and it is certainly possible that many local minima are useful in practical terms, despite not representing the smallest grammar it is possible to generate from the input.

A disadvantage to ZZ is the need to consider each possible addition to the current set of constituents before making a selection, to discover the next node of the lattice in the path which leads to the locally-minimal encoding of interest. Without prior knowledge of which addition is locally optimal, a selection cannot be made, yet computation of the encoding itself via Minimal Grammar Parsing (Carrascosa et al., 2011) is expensive, and this complexity is scaled by the number of candidate constituents which must be considered. For an input containing n candidates, for any current set of chosen constituents C there are $n - |C|$ encodings which must be computed before the next addition to C is known. This makes exploration of the space proportional to both $|C|$ and the length of the input, since the process of MGP mandates at minimum that each node in the graph, representing each symbol in the input, be considered. An ideal solution might quickly predict the size of each encoding given the possible choices, without the need to compute them entirely. In the absence of such foresight, it is possible an approximation of these sizes may be arrived at by some method less complex in computational terms than Minimal Grammar Parsing.

Discovery of such a method is not only desirable as a step towards a computationally cheap algorithm which may rival or better ZZ in the production of highly compact grammars, but also to enable the production of more complex grammars at a computational cost similar to that currently required to generate a straight-line grammar. This is especially important given the scaling which the addition of a dimension of rule modifiers introduces to grammar construction complexity. As discussed in Chapter 6, the space of all possible grammars allowing rule modification is $2^{n_c} \cdot 2^{n_m}$ where n_c is the number of candidate constituents and n_m is the number of candidate rule modifiers, compared to a space of 2^{n_c} for a grammar without modifiers. Since the value of 2^{n_c} is likely to be considerable, scaling by even a small value of 2^{n_m} is likely to result in a prohibitive search space. As shown by Carrascosa et al. (2011), ZZ is capable of exploring a space of 2^{n_c} with complexity $O(n^5 \cdot m^2)$, where n is the length of the input and m is the number of candidate constituents in the input. Introducing another dimension composed of candidate rule modifiers requires a grammar to be constructed from its candidate constituents for each modifier considered, in the simplest case. Potentially, a transform may exist for each substring which allows a complete cross-equivalence; where there are $O(n^2)$ possible substrings, there is the potential for $O(n^4)$ transforms, resulting in a complexity of approximately $O(n^9 \cdot m^2)$. Despite the considerable improvement ZZ offers over a naïve exhaustive exploration of this space, in order to allow a practical constructor to be built for grammars which allow rule modification, further improvement is clearly needed.

This chapter presents an alternative method of selecting candidate constituents, based on *Occurrence Optimisation* and the ZZ search algorithm (Carrascosa et al., 2011), which allows a strong reduction in the constituent search space to occur during grammar construction. Its novelty lies in the dynamic approximation of constituent suitability in a manner which is computationally simple. An improved construction time is possible not because the method reduces the functional complexity of constituent selection, but because it simplifies the operation of evaluting a constituent's fit prior to selection. By approximating the suitability of a constituent in a given solution, the expensive use of Minimal Grammar Parsing as a prior to adding to C during construction is avoided, and only conducted for the most promising candidates to verify their actual suitability as they are added.

The process of approximating constituent gain is well known; IRR schemes (Carrascosa et al., 2011) implement a score function such as ML (maximal length), MF (most frequent), or MC (maximal compression in an attempt to generate a prior order of selection which will most benefit the grammar overall. However, these functions calculate gain based only on the potential replacements a given constituent might offer, such $g = (i-1)l - i - l - 1$ where i is the number of repeat instances and l is the length of each repeat. They do not account for any reduction in gain due to the presence of existing constituents which may be more optimally occupying elements also associated with the new candidate, at any stage of construction. ZZ, however, adopts an opposite approach, and evaluates the overall reduction each constituent actually offers at each stage, greedily selecting a most optimal choice based on this information. Although ML, MF and MC are simple to calculate and thus fast to compute, ZZ requires an MGP for each candidate prior to selection, and computation is comparatively slow. The novel method presented here attempts to leverage the advantages of both approaches, by making use of prior knowledge regarding candidate constituents and approximating the MGP process, allowing the latter step to be computed simply.

The following sections describe the new approach, discuss its properties, and present algorithms enabling its implementation.

8.2 Approximation of Constituent Gain

A simple but effective method of constituent selection may be found by approximating the result of evaluating a single candidate's addition to an existing grammar, which would otherwise require an expensive Minimal Grammar Parse. If no edges overlapped within the parse graph, it would be possible to immediately identify the most compressive constituent and add this to the set of constituents chosen to produce the smallest possible grammar. When two or more edges overlap, this introduces a branch within the tree of possible edge choices, where each branch has a potentially reduced set of candidate constituents; such reduction in candidate space is directly related to the reduced potential of the available candidates to compress, due to the unavailability of associated edges which may previously have provided them with a sufficient advantage. As such, stepwise selection of a suitable candidate may be considered not only a problem of (a) choosing the most compressive candidate from all available constituents, but also of (b) calculating the remaining compression possible given the candidate's edges in the current solution, and (c) calculating the compression which will be lost from future candidates given the edges belonging to the current candidate which will become part of the final solution.

Since a practical evaluation of observation (c) above requires knowledge which is not yet available at any given step during grammar construction, the problem may be simplified by disregarding (c), and instead testing whether removal of any currently chosen constituents offers a reduction in encoding size after each step, in the same manner as ZZ. Evaluation of (a) is simple, and may be achieved by subtracting the total cost of inclusion of the constituent in the encoding from the total gain its non-overlapping edges provide, the latter of which may be discovered by a Minimal Grammar Parse. It is worth noting that the result of (a) does not alter, and as such may be computed once for each constituent. An accurate evaluation of (b) would require a full MGP of the graph produced by inclusion of the candidate in the current constituent set. However, it may be simplified by considering each node within the parse graph which remains unused in the current solution, to which the candidate will be added, as "available for use", and as such a gain equal to the use of *that node alone*, without consideration of whether the node belongs to a candidate edge which would be "blocked" by an existing constituent edge, may be taken as the amount of gain possible for the candidate, as a rough approximation.

Such an approximation will function perfectly where no overlapping edges occur, and will deteriorate progressively as more edges become coincident, causing greater branching within the tree of possible edge selections. However, computation of this simplified representation is easy and requires very little memory. Although the scheme is unlikely to provide a huge improvement over ZZ where the proportion of candidates in the current solution is high relative to that of the final solution, in the worst case it results in a blind exploration of the candidate space, and until that point provides a greater probability of selecting a suitable candidate than ZZ, which of course does not make selections a priori. In the early stages of grammar production, it offers the opportunity to immediately select the most compressive constituents for inclusion in the grammar's encoding, and gradually degrades towards a ZZ-like exploration, allowing constituents key to the final encoding to be quickly added without excessive comparison by MGP.

8.3 Background Theory

Calculation of the actual gain it is possible to achieve through the addition of a constituent c_{next} to an existing set of constituents C is computationally expensive, requiring a single MGP for the set $\{c_{next}\} \cup C$ in the most basic case. However, it is possible to calculate approximate gain for c_{next} if certain limitations and inaccuracies are permitted. This section provides the supporting theory which is used as a basis for such an approximation.

8.3.1 Coincident Edges

When adding a constituent c_{next} to an existing constituent set C, a branch condition can occur when an edge from c_{next} coincides with an edge from an existing constituent from C, and a choice must be made between them, usually by MGP, to decide which edge – and so which rule instance – forms part of the model constructed from $\{c_{next}\} \cup C$. The interaction between many constituents is complex, and so C is considered to contain only a single constituent, c_{prev}, for this example.

In a parse graph containing just c_{next}, such as would be passed into an MGP function to generate a model for c_{next} alone, the gain which it is possible to achieve for c_{next} may be defined as follows:

$$gain(c_{next}) = (\sum_{i=0}^{n} length(e_{next}[i]) - 1) - length(c_{next}) - 1$$

Where e_{next} is the set of edges comprising c_{next}, for which $length$ returns the number of symbols belonging to an edge, n is the count of edges within c_{next}, and $length(c)$ returns the number of symbols the production rule for c contains, excluding any termination symbol.

In a parse graph containing c_{prev} and c_{next}, such as would be passed into an MGP function to generate a model for $\{c_{next}\} \cup C$, the gain which it is possible to achieve for c_{next} may be defined as follows:

$$gain(c_{next}, c_{prev}) = (\textstyle\sum_{i=0}^{n}(length(e_{next}[i]) - 1) \times e_{active_{next}}[i])$$
$$+(\textstyle\sum_{i=0}^{n}(length(e_{prev}[i]) - 1)$$
$$\times e_{active_{prev}}[i]) - length(c_{next}) - length(c_{prev}) - 2$$

Where $e_{active_c}[i]$ is a boolean value representing whether edge i of constituent c is chosen for inclusion in a grammar generated during MGP. Where two edges with index i from two constituents a and b are coincident, $e_{active_b}[i] = 1 - e_{active_a}[i]$ with either $e_{active_a}[i] = 1$ or $a_{active_b}[i] = 1$, since only one of the two coincident edges may be active in the generated model. These equations allow an edge-based calculation of constituent gain to be made for two constituents, providing no edge in either constituent coincides with more than one edge in the other.

8.3.2 Edge-based Constituent Gain

It is common to calculate the potential gain of a constituent from the length of its associated production rule, and the number of times the rule may replace a term within the grammar encoding, such as Most Compressive as discussed in 4.1.5. ML becomes inaccurate where one or more instances of a single constituent's terms overlap each other, since all instances may not be leveraged simultaneously, making the calculation $|t_i|n_i$ an overestimate. This may be remedied by using MGP to parse a graph containing only that single constituent, resulting in the set of edges which provide the *maximum coverage* of all the graph's nodes.

Once the maximum number of usable edges is known, and providing t_{next} is equal to the number of edges which will be chosen during MGP, the gain for a single edge with index i may be defined as:

$$gain(e_{next}[i]) = (length(e_{next}[i]) - 1) - ((length(c_{next}) + 1)/t_{next})$$

Where e_{next} is an array of all usable edges, and t_{next} is the count of usable edges for constituent c. The sum of all usable edge gains is then equal to the maximum possible constituent gain:

$$gain(c_{next}) = \sum_{i=0}^{n} gain(e_{next}[i])$$

Where n is the number of edges in e_{next}.

8.3.3 Node-based Constituent Gain

Assuming t_{next} is available, or a degree of error is acceptable, the unit of gain may be further decomposed as follows:

$$covg(c_{next}) = \sum_{i=0}^{n} length(e_{next})$$

$$ngain(c_{next}) = (covg(c_{next}) - |e_{next}| - length(c_{next}) - 1)/covg(c_{next})$$

Where $ngain(c_{next})$ represents any single node within the set of edges e_{next}. Effectively, the symbol-wise gain possible through addition of the constituent to $C = \emptyset$, minus the cost of including the constituent in the grammar, is spread over all usable nodes within the graph; where $covg(c_{next})$ *is* correct, $ngain(c_{next})$ will also be accurate.

Given an accurate set of edges e_{next}, representing the maximum coverage and therefore gain c_{next} can provide, an overall gain for the constituent can now be calculated from the node-based gain value p:

$$gain(c_{next} \mid p_{next}) = ngain(c_{next}) \cdot covg(c_{next})$$

8.3.4 Limitations of Node-based Constituent Gain

Because calculation of node-based gain requires prior knowledge of the number of edges which will be chosen during MGP (where all edge lengths are equal), or specific knowledge of which edges will be chosen (where edge lengths are not equal) so that a total number of nodes n can be calculated, it cannot be used to produce an accurate figure when $n \neq covg(c_{next})$.

In the best case, $covg(c_{next})$ edges will be used during MGP, and $gain(c_{next} \mid p_{next}) = gain(c_{next})$.
In the worst case, 0 edges will be used during MGP, and $gain(c_{next} \mid p_{next}) = 0$.

In the latter case, actual gain for c_{next} will be $-length(c_{next}) - 1$, since the constituent must still be included within the grammar, as a production rule with appended termination symbol, despite offering no usable edges during MGP.

Looking at this relationship, it is possible to see that $gain(c_{next} \mid p_{next})$ gradually alters from $gain(c_{next})$ to 0, in proportion to $covg(c_{next})$ as the number of nodes belonging to the constituent which are chosen during MGP decreases. When used as an approximation, it is easy to demonstrate the calculation's inaccuracy, as even removing a single node from the count of available nodes alters the value of $gain(c_{next} \mid p_{next})$ by $-ngain(c_{next})$. The unavailability of a single node may have a large effect in practice; it may cause an edge to become unavailable, altering the actual gain for the constituent by $-length(c_{next}) + 1$ (one less constituent edge, but at a gain of requiring one less rule reference within the grammar).

Examination of these factors shows that a node-based approximation of gain with respect to a single constituent c is linear in nature, whereas actual gain for c will alter in steps of $length(c) - 1$ as edge use within the graph changes. However, when the estimated number of nodes n_{est} is equal to the actual number n_{act}, the figure of estimated gain is accurate to within $length(c) + 1$, and when $n_{est} = covg(c)$ that margin becomes 0. Thus, a node-based approach to constituent gain estimation is potentially useful, albeit limited and accurate only within its bounds.

8.3.5 Limitations with respect to Coincident Edges

As discussed previously, where two or more separate constituents contain at least one coincident edge each, an interaction occurs where the gain from the previously-added constituent(s) already present in C may drop as edges belonging to the constituent currently being evaluated as an addition to C are chosen over existing edges during MGP. As such, as $gain(c_{next})$ increases, $gain(c_{existing})$ can reduce, potentially to 0. To accurately estimate the overall, actual gain $gain(C_act)$ for $C = \{c_{existing}\} \cup \{c_{next}\}$ as $gain(C_{est})$, it is necessary to include any negative change in $gain(C_{act})$ for $C = \{c_{existing}\}$ caused by the introduction of c_{next} into C. Any estimation which ignored this change is therefore inaccurate, within the following bounds:

In the best case, $gain(C_{act}) = gain(C_{est})$.

In the worst case, $gain(C_{act}) = gain(c_{next}) - gain(c_{existing})$, where only edges from c_{next} are chosen.

As such, a node-based approximation of gain with respect to all constituents in $C = \{c_{existing}\} \cup \{c_{next}\}$ and based solely on $gain(c_{next})$ can result in an overestimation of gain $gain(C_{est}) \ll gain(C_{act})$. An underestimate is not possible, because $n_{est} <= n_{act}$ where edges do not coincide, and for an edge from c_{next} to be chosen the gain produced by that constituent must, overall, be greater than that produced by selection of edges belonging to $c_{existing}$.

8.3.6 Assumption and General Properties

Assuming that constituents are added to a grammar in sequence, and the final gain for each constituent in that sequence is known in advance, any step in that sequence will see the constituents which are most compressive *overall* already present within the grammar. Thus, any constituent in the sequence will possess a gain $gain(c[i+1] < gain(c[i]) < gain(c[i-1])$, and the next most compressive constituent at index i will be $i+1$.

Actual discovery of this sequence, as discussed above, requires a full exploration of each combination of C, or prior knowledge of the binary relationship between $e_{active_{next}}$ and $e_{active_{existing}}$, pairwise, for each edge of each constituent, and for the final, optimal solution (i.e. the set of all constituents producing the smallest grammar). This exploration is expensive and impractical, and instead may be reduced to an estimated, linear relationship between $e_{active_{next}}$ and $e_{active_{existing}}$ for any step in the sequence of adding constituents to the grammar. It may be further reduced to an estimated, node-based relationship, with the following caveats:

1. Gain $ngain(c_{next})$ can be overestimated with respect to $gain(C_{act})$, since $ngain(c_{next}) >= 0$ and addition of c_{next} can cause a reduction in current gain for $c_{existing}$.

2. Gain $ngain(c_{next})$ can be overestimated with respect to $gain(c_{next act})$, since $n_{act} = k \cdot l$ where $l = length(c_{next})$ and k is the number of edges chosen during MGP, but n_{est} does not have a modulo relationship to l.

3. Gains are calculated *only* for the current constituent set C, as the most compressive combination of all candidate constituents is not known in advance.

However:

- $max(ngain(c_{next}))$ is completely accurate if $covg(c_{next})$ is based on the set of edges offering maximum coverage during MGP with $C = c_{next}$.

- All candidate constituents c_{next} are subject to the same limitations when their gains are estimated, thus gain values are correct relative to each other.

Although strongly limited, a node-based approach to selection of the next candidate to add to a given constituent set C does offer some benefits. In particular, it is linear in time complexity to compute *ngain* for each possible candidate, after which a fast sort algorithm can quickly yield an estimated most-compressive choice.

8.3.7 Expression of Linear Approximation

A linear approximation *gain* for a candidate constituent c can be computed as:

$$gain(c) = gain(c \mid p) \cdot n$$

Where n is the number of nodes in the current solution which are not yet used by any constituent edge and which are associated with the constituent c, and p indicates the use of a node-based approach to gain calculation.

Therefore:

$$0 \leq gain(c \mid p) \cdot n \leq gain(c)$$

Where $gain(c)$ is Maximum Compression (Carrascosa et al., 2011) as a function of substring occurrences and length, and $gain(c \mid p)$ is the approximate gain for a solution given an arbitrary number of existing constituents. In summary, since all candidate constituents c are affected equally by the above conditions, $gain(c \mid p)$ may be considered a reasonable approximation of $gain(c)$ which does not require an expensive MGP to be executed.

8.4 Proposed Solution

There now follows a description of grammar construction using the principles which have been discussed.

Initially, a dictionary D of repeats occuring in the input string S is constructed, as described in Chapter 6, Algorithms 1 and 2. The dictionary may be composed of exactly matching segments, or of flexible matches which rely upon a transform to provide equivalence, depending on which is most applicable to the compression of S.

Given D, directed, acyclic graphs may now be constructed for the input string S, and for each dictionary key $k \in D$, where k_n represents the n^{th} candidate production rule for the grammar G which will be constructed. To achieve a Minimal Grammar Parse, each graph may be evaluated from nodes N_{start} to N_{end}, and the edge arriving at any node N_n with the smallest cumulative cost u taken as the fastest route from $N_{<n}$ to N, and therefore chosen for inclusion in the final MGP solution. Since $u_N = \sum_{i=1}^{n} u_{N_i}$ for all nodes N_i visited during a traversal of N_{start} to N, $u_{N_{end}}$ is the minimum possible cost u of traversing the graph, and the list of nodes visited forms the MGP of the string being parsed. For a string s, this process has complexity $O(nd)$, where $n = |s|$ and d is the maximum degree of edges arriving at any N, since it is proportional to both the number of choices at each node, and the number of nodes required to represent s.

Prior to parsing, a list of active constituents may be constructed, representing all edges belonging to each active constituent. This list forms the combination of constituents currently being evaluated in the construction of G, and represents a specific node in the Lattice (Carrascosa et al., 2010) of all possible combinations. When computing a cost c for any node N in the graph, only edges which arrive at N and use either a direct path from the previous node or an active constituent edge are considered. In this manner, a single set of DAGs for S and each of its candidate constituents may be produced during initialisation, and only the active constituent list requires alteration during evaluation of any constituent combination during the construction of a minimal grammar G.

ZZ operates in "downwards" traversal mode – addition of a constituent to the current set – by considering the effect of adding a single constituent c_n to a current combination of constituents C, and the candidate c which produces a minimal $|G|$ is retained as $C = C \cup \{c\}$. Assuming that no "upwards" traversals are required during the construction of G – that no constituents may be removed to minimise $|G|$ – the number of MGPs required to produce a grammar G with final constituent set of size $l = |C_{final}|$ is $|C_{final}| \cdot |D_{keys}|$ since all keys in D may have been considered when adding a single constituent from D_{keys} to C_{final}, and this must occur at minimum $|C_{final}|$ times. Parsing each graph is a relatively expensive process, and the solution proposed here improves on ZZ's traversal of the Lattice using three specific changes:

1. Use of $u_i = MGP(C \cup \{c_i\})$ for each $c_i \in D_{keys}$ is replaced with a less expensive gain approximation g, representing the total number of nodes which may be skipped in the parse graph following the inclusion of c_i.
2. The list of possible gains $g_i = g(c_i)$ for each $c_i \notin C$ is sorted prior to considering $MGP(C \cup \{c_i\})$, to approximate consideration of a "best" constituent first, with $g(c_i) \geq g(c_{i+1})$.
3. During any $MGP(C)$, the set of constituents which are *actually used during parsing* are returned, and the resulting computation of $|G|$ is based only upon these.

Computing an order of constituents to evaluate in this manner attempts to ensure that the acceleration of a downwards traversal of the Lattice does not result in inclusion of constituents which are not useful in minimising $|G|$, and aids in preventing the greediness of the algorithm from causing a non-optimal solution to be constructed. Computation of $|G|$ based only on used constituents allows the smallest encoding given $MGP(C)$ to be discovered, and any completely unused constituents to be discarded since they are not relevant to the current solution.

8.4.1 Algorithm Pseudocode

Overview

Pseudocode which models the process described above is shown in Algorithms 4 and 5. Firstly, the maximum possible gain which may be obtained from the inclusion of the ith constituent in a grammar

G is recorded, by generating a grammar containing only that constituent, and calculating the difference between its size and the length of the original input *S*. These gains may be used to indicate the best-case potential of each constituent, and guide construction of the actual grammar. Algorithm 4 illustrates this step.

Once the maximum possible gain of each constituent is known, the order in which they are likely to be most beneficial to a grammar *G* containing a given combination of constituents may be derived using the bitmask-based approximation described above, and candidates may be added to *G* based upon this ordering. Algorithm 5 illustrates this process. The following steps describe the general scheme followed by Algorithm 5, and are provided as clarification of the purpose of the groups of statements which the algorithm contains. Indentation is used here to signify that a block of operations is contained within a loop, the continuation of which is loosely specified by the statement immediately above:

```
Repeat until no improvement to $|G|$ has occurred during an entire pass:
    Create a sorted list of approximate gains, gainsSorted (one for each constituent).
    Let $i$ equal the index of the first constituent in gainsSorted.
    Repeat while no improvement to $|G|$ has occurred, and constituents remain to be tested:
        Add constituent $i$ from gainsSorted to $G$, and compute $|G|$.
        If $|G|$ has reduced, keep the new constituent in $G$.
        Otherwise, increment $i$ to operate on the next constituent in gainsSorted.
    Let $j$ equal the index of the first constituent in $G$.
    Repeat while an improvement to $|G|$ has occurred:
        Remove constituent $j$ from $G$, and compute $|G|$.
        If $|G|$ has reduced, continue without the constituent in $G$.
        Otherwise, increment $j$ to operate on the next constituent in $G$.
```

Process

This section presents the pseudocode for Algorithms 4 and 5, and describes their operation in greater detail. Pseudocode for the creation of empty bitmask arrays is also provided in Algorithm 3, to simplify the expression of those algorithms.

Initially, a parse graph set *P* is constructed for *S* and each candidate constituent, containing edges for *S* and all available constituents which are to be considered for inclusion in *G* (with the set of all possible constituents denoted by C_{all}). Within each graph, each edge contains an associated constituent number, an edge instance number, and an edge cost. Edge costs are set at 1 for all edges except those requiring a transform, which are instead set at 2, since they require a transform reference symbol in addition to their production rule reference symbol during encoding, as discussed in Chapter 6, Section 6.4.2.

As an initial step, each constituent c is individually activated, and a Minimal Grammar Parse of P given $C = \{c_i\}$ is performed, beginning with the graph representing S. The edges chosen during $MGP(P, C)$ which belong to c_i are examined, and the total number of nodes avoided through the use of these edges is recorded as the gain g_i for that constituent c_i. These values represent the maximum gain it is possible to achieve through the addition of c_i to C, but do not take into account the cost of including c_i within the encoding G as a production rule. Algorithm 4 shows how this may be achieved given the graphs P and constituents C for which gain values are to be computed.

Algorithm 3 createBitmasks(): Create a blank bitmask array for S, and one for each constituent

```
Require: $lengthS, constituentLengths$
Ensure: $b_S, b_{Cons}$
 1: $b_S \leftarrow false[lengthS]$                               ▷ Create an empty bitmask for S
 2: for $i \leftarrow 1$ to $length(constituentLengths)$ do
 3:     $b_{Cons}[i] \leftarrow false[constituentLengths[i]]$    ▷ Create an empty bitmask for each constituent
 4: end for
```

Algorithm 4 computeMaxGains(): Compute the maximum possible gain for each constituent in C

```
Require: $P, C$
Ensure: $maxGains$, (array of size $|C|$)
 1: for $i \leftarrow 1$ to $length(C)$ do
 2:     $maxGains[i] \leftarrow 0$
 3:     $cost, usedEdges \leftarrow MGP(P, \{C[i]\})$
 4:     for $edge \leftarrow usedEdges$ do
 5:         if $edge.constituentNumber = i$ then
 6:             $maxGains[i] += edge.endNode - edge.startNode - 1$
 7:         end if
 8:     end for
 9: end for
```

Once the maximum possible gain of each constituent is known, the search for a combination of constituents C which will produce a compact encoding of G may begin. Algorithm 5 illustrates this process.

Algorithm 5 computeCompactC(): Compute a constituent set C belonging to compact grammar encoding G

```
Require: $P, C_{all}, S, constituentLengths$
Ensure: $C$ (combination of constituents from $C_{all}$ which produce a compact encoding $G$)
 1: $maxGains \leftarrow computeMaxGains(P, C)$
 2: $numRestarts \leftarrow 0$
 3: $C \leftarrow \emptyset$
 4: $cost \leftarrow Inf$
 5: $b_S, b_{Cons} \leftarrow createBitmasks(length(S), constituentLengths)$
 6: while $numRestarts < 2$ do
 7:     $gainsSorted, gainsOrder \leftarrow sort($
                $computeApproximateGains(C \uplus C_{all}, C, b_S, b_{Cons}, maxGains,$
                $length(S), constituentLengths), order = descending)$
 8:     $i \leftarrow 1$
 9:     while $repeat$ do
10:         $repeat \leftarrow false$
11:         $c_{target} \leftarrow gainsOrder[i]$
12:         $C_{add} \leftarrow C \cup \{c_{target}\}$
13:         $cost_{add}, usedEdges \leftarrow MGP(P, C_{add})$
14:         if $cost_{add} < cost$ then                  ▷ If an improvement occurred
15:             $C \leftarrow C_{add}$
16:             $cost \leftarrow cost_{add}$
17:             $b_S, b_{Cons} \leftarrow addConstituentToMask(b_S, b_{Cons}, C, c_{target})$
18:             $numRestarts \leftarrow 0$
19:         else                                         ▷ If an improvement did not occur
20:             if $i < length(gainsSorted)$ then        ▷ If there are still candidates to
                                                           consider in $gainsSorted$
21:                 $i++$
22:                 $repeat \leftarrow true$
23:             else
24:                 $numRestarts++$
25:             end if
26:         end if
27:     end while
```

(Algorithm 5 continued)

```
28:     while $improvement$ do
29:         $improvement \leftarrow false$
30:         $C_{del}, cost_{del} \leftarrow ZZRemove(P, C)$        ▷ Try a ZZ-style removal of a constituent
                                                                      in C to minimise $cost$
31:         if $cost_{del} < cost$ then
32:             $C \leftarrow C_{del}$
33:             $cost \leftarrow cost_{del}$
34:             $b_S, b_{Cons} \leftarrow createBitmasks(length(S), constituentLengths)$
35:             $usedCons \leftarrow membersOf(C)$
36:             for $j \leftarrow 1$ to $length(usedCons)$ do
37:                 $c_{target} \leftarrow usedCons[j]$        ▷ Store the number of current constituent
                                                          to add to bitmasks $b_S$ & $b_{Cons}$
38:                 $b_S, b_{Cons} \leftarrow addConstituentToMask(b_S, b_{Cons}, C, c_{target})$
39:             end for
40:             $improvement \leftarrow true$
41:         end if
42:     end while
43: end while
```

At the beginning of each iteration, a list of approximate gains which may be expected due to the addition of each c_i to C is computed as shown in Algorithm 7, and sorted in descending order as shown after the first *while* statement in Algorithm 5. Using this ordering, a Minimal Grammar Parse using $C \cup \{c_{target}\}$ is performed, and the constituent c_{target} only retained and added to C if the resulting overall encoding size $|G|$ is smaller than any encoding yet seen.

If the constituent does not meet the above criteron, it is rejected, and the next constituent in the ordered list considered for addition to C by incrementing the index i. When the list is exhausted, an attempt is made to further reduce the encoding size $|G|$ by removing constituents currently in C in the same manner as ZZ (represented by the function call $ZZRemove(P, C)$ in Algorithm 5). In testing whether a removal reduces the model size, each $c_i \in C$ is removed in turn, and the combination with $|G_{C_{del}}| < |G_C|$ producing the minimal value of $|G|$ is retained; where all $|G_{C_{del}}| \geq |G_C)|$, no alteration to C occurs.

These attempts continue until no improvement in $|G|$ is possible, at which point execution returns to the *while* statement at line 9, and a new overall iteration begins.

The search for a C which minimises $|G|$ continues until a specific condition is met, that of $numRestarts = 2$. When no improvement to $|G|$ may be made within an iteration, during which time all constituents which remain to be added to C have been evaluated for their potential to minimise encoding length, *numRestarts* is incremented. Whenever an improvement occurs, *numRestarts* is reset to 0. Therefore, once this value equals 2 there are no constituents to add to C which are useful in reducing the encoding length, and no removal of constituents from C can improve $|G|$. At this point, the algorithm terminates.

8.4.2 Approximation of Constituent Gain

Fast construction of a grammar based on constituent selection may be said to depend upon prior knowledge of which constituents will be useful in the minimisation of $|G|$, and so allow selection of those which are required to form a compact final encoding. ZZ relies on testing each potential addition to an existing constituent set C before including it. Although this is an expensive computation given significantly large or repetitive inputs, the approach benefits from being able to observe the actual effect of a constituent *on the current encoding*, or indeed the effect of removing an existing constituent from the encoding in the case of an upwards lattice traversal, and this advantage allows ZZ to improve upon greedy approaches such as IRR-MC.

A key improvement offered by this study's method is the ability to evaluate the effect of a constituent c on an existing encoding through the use of a far simpler computation, at the expense of arriving only at an approximation of the constituent's ability to offer an improvement to $|G|$. To achieve this, *only* the gain which is approximately available in the current solution, given the use of any edge belonging to the new candidate constituent c, is considered as a basis for its candidacy.

This is made possible by treating each node associated with an edge belonging to c as providing a proportion of the overall possible gain c can provide, in a solution containing no other constituents, i.e. where c is maximally used. This results in a gain per node of $x = g/n$, where g is the total gain reported by Algorithm 4 for c_i, and n is the total number of unique nodes associated with the edges of constituent c.

A bitmask $b_{current}$ of all nodes currently used by any constituent C in a given solution may be maintained during grammar construction, and a bitmask $b_{candidate}$ quickly generated for the c being considered for inclusion in C. Computing $b_{new} = (NOT\ b_{current})\ AND\ b_{candidate}$ is trivial, and the sum $t = sum(b_{new})$ can be taken as an approximate indication of the number of times the node-wise gain x might be applied in the current solution to minimise $|G|$. Thus, an approximate gain for c may be produced by calculating $g_c = xt$, and g_c for all $c \notin C$ used to discriminate between likely useful and non-useful candidate constituents, *a priori*.

Algorithm 6 is included below to demonstrate the addition of bits for the edges of a constituent q to a bitmask set belonging to an existing constituent combination C. It is a utility function for Algorithms 5 and 7, and produces a bitmask set showing where the edges of q occur within the current solution's parse graphs when superimposed upon the existing bitmasks.

The following is a description of the process shown in Algorithm 7, to clarify the simplicity of its operation. First, a set of zeroed masks are constructed for S and for each constituent in C, the latter with dimensions $length(c_i \in C)$. Each bit within these masks represents a node within the current parse graphs. As discussed in regard to Algorithm 6, when considering the effect of adding a constituent q to C, the nodes which its edges allow to be avoided may be set to *true* within each mask, for S and each constituent in C, to indicate the positions in which these edges exist. The zerored masks may conveniently be passed into the function described by Algorithm 6, and their bits set as required by the existence of the edges belonging to q.

Each constituent q from the set Q is then individually processed by Algorithm 7 to discover an approximate gain for $C \cup \{q\}$, with the resulting value stored in the array *approxGains*. This is achieved by first fetching a bitmask set $(b_{tmpS}, b_{tmpCons})$ for q, as described above, and counting the number of nodes associated with q which are not yet associated with any other constituent in C. Discovery of the available nodes may be made using a simple bitwise *AND* operation for each mask, which is computationally cheap in comparison to a call to $MGP(P, C \cup \{q\})$. Given the total nodes associated with q, and the number of these which are unused by other constituents in C, the proportion of nodes

Algorithm 6 addConstituentToMask(): activates bits within the input bitmask set b_{inS}, b_{inCons} (associated with the constituent set C) where an edge from a constituent q is associated with any node in the parse graphs for C

```
Require: $b_{inS}, b_{inCons}, C, q$
Ensure: $b_{outS}, b_{outCons}$
 1: $b_{outS} \leftarrow b_{inS}$
 2: $b_{outCons} \leftarrow b_{inCons}$
 3: $constituentEdges \leftarrow getConstituentEdges(q)$    ▷ Each element of $constituentEdges$ contains a start and end node index, and the number of the constituent the edge occurs within
 4: for $edge$ in $constituentEdges$ do
 5:     if $edge.occursIn = 0$ then    ▷ If this constituent occurs within S
 6:         $b_{outS}[edge.startNode : edge.endNode] \leftarrow true$
 7:     else if $edge.occursIn \in C$ then
 8:         $b_{outCons}[edge.occursIn][edge.startNode : edge.endNode] \leftarrow true$
 9:     end if
10: end for
```

Algorithm 7 computeApproximateGains(): computes a list of approximate gains for a set of constituents Q based on bitmasks b_S, b_{Cons} for the current constituent set, C

```
Require: $Q, C, b_S, b_{Cons}, maxGains, lengthS, constituentLengths$
Ensure: $approxGains$ (a list of approximate gains of length $|Q|$ for each $q \in Q$)
 1: $b_{emptyS}, b_{emptyCons} \leftarrow createBitmasks(lengthS, constituentLengths)$
 2: $unusedCons \leftarrow membersOf(Q)$
 3: for $i \leftarrow 1$ to $length(unusedCons)$ do
 4:     $q \leftarrow unusedCons[i]$    ▷ Store the number of current constituent for which to calculate gain
 5:     $b_{tmpS}, b_{tmpCons} \leftarrow addConstituentToMask(b_{emptyS}, b_{emptyCons}, C, q)$
 6:     $totalNodes \leftarrow sum(b_{tmpS})$
 7:     $availableNodes \leftarrow sum(b_{tmpS}$ AND $(1 - b_S))$
 8:     for $c \leftarrow 1$ to $length(b_{Cons})$ do
 9:         $totalNodes += sum(b_{tmpCons}[c])$
10:         $availableNodes += sum(b_{tmpCons}[c]$ AND $(1 - b_{Cons}[c]))$
11:     end for
12:     $approxGains[i] \leftarrow (availableNodes / totalNodes) \cdot maxGains[q]$
13: end for
```

which remain available to provide potential gain is calculated, and the previously computed maximum gain *maxGains* scaled by this amount to produce an estimation of the gain which q may offer to the current solution C.

There are several assumptions made by this approximation, due to the high level of simplification it takes towards the problem. However, since these are applied to every constituent during calculation of their approximate gain, the relationship between them in the current solution C is generally maintained, and as such the resulting values provide a useful practical approximation of the actual gain which may be expected from $C \cup \{q\}$.

8.5 Failure Modes

This method of estimating constituent edge use, given that it is a simplistic approximation, is susceptible to failure under certain conditions. Perhaps the most obvious is within a solution where the edges of the candidate being considered overlap those of an existing, more optimal constituent. Here, it will produce a positive estimation of gain, whereas the existing constituent's edges will always be selected during Minimal Grammar Parsing, and the estimation should in fact be non-positive. If a second candidate exists whose edges do not overlap those of any existing constituent, but the candidate's maximum gain is low, it is possible that it will be initially ignored as a suitable constituent since it appears another, better alternative exists. This particular failure is alleviated, but not fully addressed, by the use of MGP prior to final inclusion of the candidate in the grammar as shown in Algorithm 5. Since MGP returns the actual, overall gain resulting from the candidate's addition, testing whether an improvement in encoding length occurred prevents any failures in estimation from affecting the set of constituents chosen, and instead only those with a genuine positive gain value will be retained.

However, it is possible for a bitmask to become "saturated", either because a high proportion of true values exist which overlap with the edges of all remaining candidate constituents, or because all positions have been occupied by an existing edge, and none remain by which a candidate's gain may be estimated. In such circumstances, the existence of a candidate which reduces the encoding length is still possible, and such a candidate will provide a gain which exceeds that of any existing combination of constituents it will replace. Ideally, this circumstance would not occur – if approximation of gain were accurate, it would already have been selected over any less-optimal constituents – but as the algorithm only observes a change in grammar encoding length when adding a constituent with non-zero gain, it is not improbable that the better candidate may be considered later in the construction process.

Given n positions within the bitmask, as the number of occupied positions approaches n there are progressively less opportunities for a candidate's edges to achieve an estimated positive gain, and so the range of possible values shrinks and causes a reduction in the ability to discriminate between candidates, ultimately resulting in all candidates being considered of equal value. At this point, it is impossible to sort the remaining constituents by their gains, and so the first candidate which produces a reduction in encoding length following MGP will be added to the grammar at each iteration. Optimal selection of a candidate is therefore impossible, and further approximation will fail.

How and when this occurs depends on the structure of the input data, and on how accurate the approximation has been at any given point in the construction process. Some inputs may produce candidate constituents which are largely disjoint from each other, making it easy for the algorithm to discriminate between them. Where gains have been accurately estimated, the optimal set of constituents will have been selected before bitmask saturation begins to degrade the algorithm's ability to usefully estimate them, preventing any adverse effect. It is beyond the scope of this work to examine such failure modes in depth, but a study of their occurrence, and research into their prevention, would be useful as future work.

8.6 Summary

In this chapter, a method was presented by which the potential gain of candidate constituents may be estimated during grammar construction, based upon the existing constituent edges which are in use at any stage of construction, and the positions which remain unused and may be leveraged by a candidate's edges. Theory was outlined showing that the method produces a linear approximation of a candidate's gain, in constrast to the binary selection of edges which result in the actual gain produced by a constituent during Minimal Grammar Parsing. However, the bitmask-based approach to its calculation is far less complex to compute than MGP performed on a graph representing a grammar's elements, and so it enables the number of MGPs required during construction of the grammar to be massively reduced by replacing their function with a simple calculation which can be computed in a shorter time. As such, the method provides an improvement over the use of ZZ to traverse the lattice of candidate constituents, and allows an approximately-optimal ordering of candidates to occur from which the most immediately beneficial option may be selected, effectively limiting the space of lattice nodes which must be considered prior to selection of a locally-ideal constituent combination to only those which are approximately ideal given the current solution. In this manner, compact grammars can be quickly constructed using a fraction of the expensive MGPs required by an unmodified ZZ traversal, and in a greatly reduced time.

In the following chapter, the performance of an implementation of this approach to the construction of grammars from real-world data will be investigated, with the aim of demonstrating the validity of the theory behind the approach, and its relevance to practical applications.

The true method of knowledge is experiment.

William Blake, 1788

9 An improved method of Lattice Traversal: Experiments & Results

9.1 Introduction

Any approximation of global optimisation, i.e. any locally-optimal scheme, is by definition likely to produce sub-optimal results. The way in which the terminal state is reached depends on the specifics of the problem being addressed, and of the scheme in operation. Each such scheme may be said to possess particular characteristics which determine its ability and performance, for example the optimum accuracy which may be expected in practice, and the classes of input which are known to cause failures and degraded performance.

In this chapter, several experiments are presented which explore the performance of the improved lattice traversal method discussed in Chapter 8, designed to highlight the improvement it provides over ZZ and discover the circumstances under which it operates poorly. Of particular interest is the manner in which it operates in the worst case: should its ability remain at least equivalent to ZZ in such instances, any benefit which it provides at other times may safely be leveraged without resorting to ZZ itself as a fallback process. Performance is explored not only on music corpora, as the variety of data which is of primary interest in this thesis, but also on standard DNA and linguistic corpora, the use of which represents best practice in the evaluation of the performance of grammar-based compression algorithms (Benz & Kötzing, 2013; Carrascosa et al., 2011; Siyari & Gallé, 2017).

In these experiments, the new method will firstly be used to construct grammars for inputs from standard corpora, to provide a direct comparison against the results of existing studies. Pieces from the corpus of musical scores gathered specifically for this study will then be used as inputs, to examine the differences in the method's response to musical and non-musical input types. Finally, a detailed analysis of the construction process itself given specific inputs is provided, to show how the scheme achieves its improvement in practice, and identify input types for which it is particularly appropriate.

Given the high construction times for particularly large or complex inputs, all relevant experiments are conducted on collections which are ordered by approximate complexity, so that the greatest number of inputs may be processed in the available time. Where inputs are from standard corpora such as the Canterbury Corpus (R. Arnold & Bell, 1997) or a collection of DNA sequences (Grumbach & Tahi, 1994), they are processed in order of ascending file size. Where inputs are from the corpus of digital scores described in Chapter 3, they are processed in ascending order of ZZ-based grammar construction time, in the same manner as the experiments presented in Chapter 5. The proportion of the collection processed in each experiment is recorded in its associated results section.

9.1.1 The Importance of Candidate Ordering

It is important to note that a direct comparison between grammar construction methods based on an insufficient number of samples and encoding size alone is not likely to be fully indicative of performance. ZZ itself is deterministic, since it consistently selects the locally optimal choice for any given position in the lattice, and does not contain any randomisation of its behaviour, rendering its output entirely repeatable on separate executions.

However, it is possible to implement a ZZ-based grammar constructor in different ways; candidate constituents may be stored or selected in a variety of different orderings, for example by the position of the first substring instance in the input string, by the length of the first substring, or, in a transform-enabled system, by the type of their associated transforms. This ordering has a marked effect on the final encoding – given a set of candidates which all produce the same reduction in grammar size, only the first ($|G_{next}| < |G_{curr}|$) or last ($|G_{next}| \leq |G_{curr}|$) candidate will be chosen for inclusion at any given iteration, and every candidate has the potential to "block" another, resulting in a different path being chosen during lattice traversal. Therefore, if candidates are trialled in a different order, a different encoding is likely to result, often with a very different length.

Due to this, it is important not to immediately equate a smaller output encoding size with a more optimal compressor. Over a large sample, it should be possible to observe a general trend in compression strength, but, as discussed previously, grammar construction for large inputs is inherently expensive and so it is unlikely a sufficient sample size is achievable for a single study if a good distribution of large inputs are to be included. Instead, it is perhaps more indicative to examine the *general* level of compression which a method achieves for such inputs – it is known that ZZ produces consistently smaller grammars than IRR schemes, and so where the method presented in Chapter 8 is capable of improving on the results of IRR, and encoding size does not differ significantly from that of ZZ-generated grammars, it is probable that the new method is performing approximately as effectively as ZZ itself. Equally, where the new method produces a smaller grammar, this does not necessarily indicate a stronger ability to compress, but rather the reaching of a different locally-optimum node in the lattice. The primary intention of the technique presented in Chapter 8 is to provide a similar level of compression to ZZ, but to do so in a more time-efficient manner.

A definitive empirical approach to comparing the ability of ZZ against that of the method this study presents would be exhaustive: it is the *inability* of IRR schemes to reach nodes in the lattice which are as optimal as those arrived at by ZZ which indicates a failure in IRR to optimise occurrences, as identified by Carrascosa. Given that a different ordering of candidate constituents can result in a different path through the lattice being chosen, all possible orderings could be tested and the nodes reached recorded. Where our method can be shown to arrive at the same set of nodes, it may be said that compression potential is equal to that of ZZ. Even on a small input with n candidate constituents, there are $n!$ possible constituent permutations, making such an exhaustive experiment computationally costly to run. For this reason, only an approximation is sought within the following experiments between ZZ and the presented method, alongside and improvement in comparison with IRR schemes, as a general indication of performance equality.

9.1.2 Applications

Compression of a given input may be considered a primary application of a grammar construction algorithm. From the compressed representation, other applications become possible, such as calculation of a compression-based distance between inputs. As such, compression effectiveness and efficiency are aspects which are important to the evaluation of any new or modified compression scheme. This section explores the effect of the improved lattice traversal method presented in Chapter 8 on the compression of standard and musical corpora, both when constructing grammars containing non-altering production rules, and when constructing grammars whose rules may be modified during expansion.

9.2 Experiment 1: Grammars constructed from Standard Corpora

In this experiment, the method of grammar construction detailed in Chapter 8 is used to generate grammars for selected items from the Canterbury Corpus (R. Arnold & Bell, 1997) and a collection of DNA sequences (Grumbach & Tahi, 1994). Sizes of the resulting encodings are compared to those produced by ZZ and IRR-MC (Carrascosa et al., 2011, 2012), and the number of operations required to produce a grammar with an equal number of constituents is compared for both methods.

9.2.1 Purpose

The experiment is specifically designed to demonstrate that the dynamic ordering of constituents by approximate gain allows grammars to be constructed significantly faster than when using ZZ, given real-world data. The process of computation of an approximate gain for a given constituent is clearly not a free one, although it is not as expensive as performing a Minimal Grammar Parse. The experiment is also designed to show that, despite this, fewer operations overall of any type are required to produce an approximately equivalent grammar, i.e. one which results from arrival at a node in the lattice which is locally optimal and reachable by ZZ search given a particular constituent ordering.

9.2.2 Method

Given the time complexity of grammar construction for large inputs, items are selected from the Canterbury and DNA Corpora based on their storage size, in ascending order, so that as many inputs may be processed as possible given the available experimental time. The order of computation may be seen in Table 9.1.

For each item, all data is read into an array, and represented as a sequence of integers. The sequence is then passed as input into an implementation of the algorithm detailed in this study. During construction, a count is kept of how many MGP operations are performed, and how many times a bitmask is updated and applied, for instance to approximate the gain of adding a new constituent to an existing combination. Record is also kept of the number of MGPs and current encoding size each time a more compact encoding is discovered. Each reduction in model size is the result of adding a new constituent to the grammar, or removing a constituent which is no longer useful. Construction continues until addition or removal of any constituent from the current set of constituents provides no decrease in encoding size.

This process is repeated for all inputs tested, and the resulting encoding sizes and operation counts analysed and directly compared to their ZZ-derived equivalent. Encoding sizes for ZZ-generated gram-

Canterbury Corpus		DNA corpus	
File	Bytes	File	Bytes
grammar.lsp	3721	humdyst	38770
xargs.1	4227	humprtb	56737
fields.c	11150	humhdab	58864
cp.html	24603	humghcs	66495
sum	38240	humhbb	73308
asyoulik.txt	125179	mtpacga	100314
alice29.txt	152089	chmpxx	121024
lcet10.txt	426754	chntxx	155844
plrabn12.txt	481861	mpomtcg	186609
ptt5	513216	vaccg	191737
kennedy.xls	1029744	hehcmv	229354

Table 9.1: Items from the Canterbury and DNA Corpora, ordered by file size in bytes – processing in this order allows the largest number of items to be considered in the shortest time.

mars are drawn from existing studies (Carrascosa et al., 2011, 2012), and where a study reports an encoding size as a percentage of a quantified IRR-MC encoding, the most compact possible encoding is calculated from these values.

9.2.3 Results

DNA Corpus

Grammars were generated for three items from the DNA corpus; Table 9.2 contains the results, and relevant metrics computed from the process or from the associated studies. The minimum number of MGPs required to produce an equivalent grammar using ZZ is approximated by cn, where c is the number of constituents used in the encoding and n is the total number of candidate constituents during construction. This approximately represents optimal construction, and assumes no occurrence optimisation is attempted or required (i.e. construction was completed using only "add" steps by ZZ). Grammar sizes are reported in number of symbols.

Encoding length at each observed reduction during construction was plotted against the number of MGPs for each piece, to show the construction rate achieved, as may be seen in Figures 9.1 – 9.3. The rate at which the equivalent grammar is constructed by ZZ is also plotted on each figure, to provide a direct indication of the improvement possible through the use of the bitmask-based method.

Sequence:	humdyst	humprtb	humhdab
$\lvert G \rvert$	10,050	13,667	13,980
Number of constituents chosen	403 of 29,986	706 of 60,662	783 of 71,252
Number of MGPs	359,912	1,820,940	3,340,902
Time per MGP (s)	0.33	1.858	2.019
Number of Bitmasks	17,813,688	60,211,460	80,983,229
Time per Bitmask (s)	1.57×10^{-4}	0.0123	2.95×10^{-4}
Total time (s)	122,755	4,164,184	6,768,176
$\lvert G_{ZZ} \rvert$	10,078 (-8.93%)	13,658 (-8.27%)	13,993 (-8.7%)
Number of MGPs (min)	12,084,358	42,827,372	55,790,316
Total time (min, s)	3,991,500	79,573,257	112,623,981
$\lvert G_{IRR-MC} \rvert$	11,066	14,890	15,327
MGP ratio (this vs. ZZ)	$\underline{0.03}$	$\underline{0.043}$	$\underline{0.06}$
Time ratio (this vs. ZZ)	$\underline{0.031}$	$\underline{0.052}$	$\underline{0.061}$

Table 9.2: The result of generating grammars encoding items from the DNA corpus using the method described in the previous chapter. Encoding sizes for ZZ- and IRR-MC-derived models are taken from studies by Carrascosa et al., with the stated improvement for ZZ over IRR-MC given as a percentage in brackets, following the calculated ZZ model size.

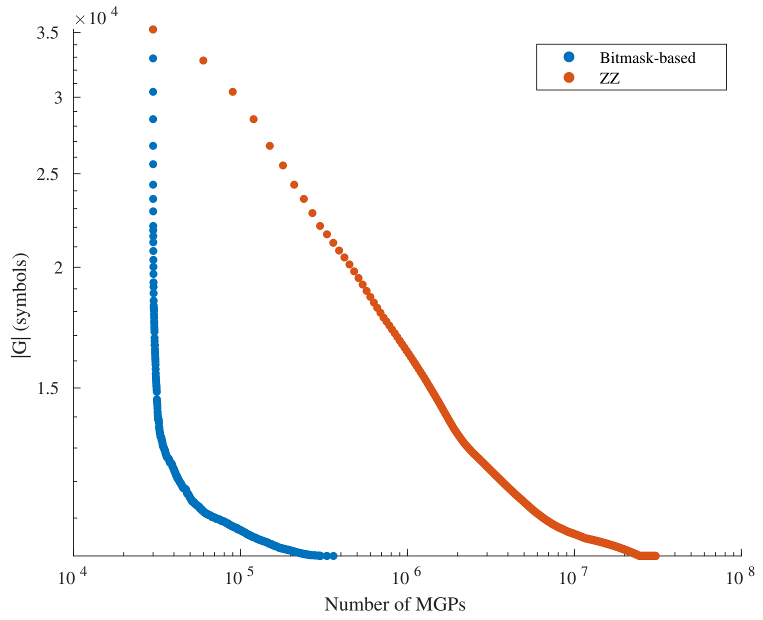


Figure 9.1: Number of MGPs to encoding length observed during construction, for *humdyst* from the DNA corpus, shown for the bitmask-based method and standard ZZ. Both axes are plotted logarithmically to highlight the difference in performance.

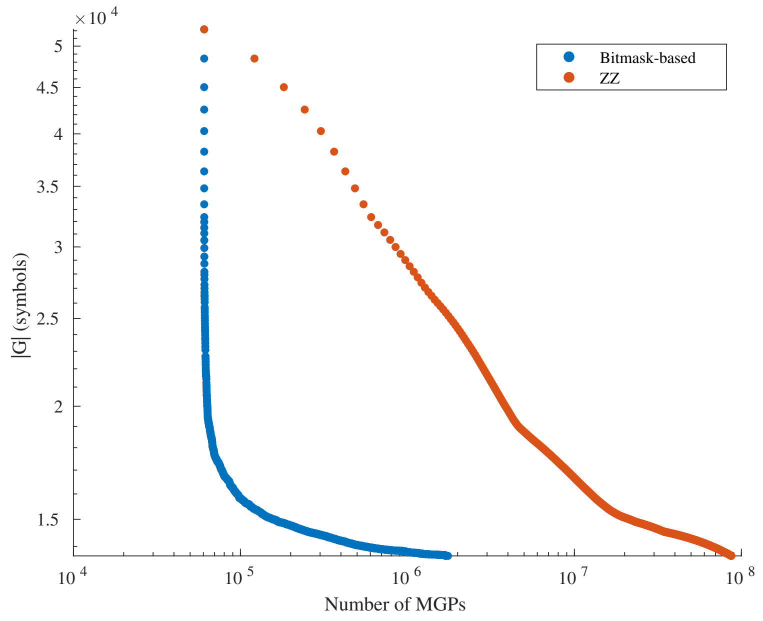


Figure 9.2: Number of MGPs to encoding length observed during construction, for *humprtb* from the DNA corpus, shown for the bitmask-based method and standard ZZ. Both axes are plotted logarithmically to highlight the difference in performance.

9.2.4 Analysis

From the results, it may be stated that the new method generates models of comparable size to those created by ZZ, and significantly smaller than those created by IRR-MC. The newly-generated grammars are not equal to their ZZ counterparts; for example, the ratio of encoding size of the new method to ZZ for *humdyst*and *humhdab* is < 0.998 and < 0.992 respectively, but for *humprtb* it is less than < 1.001. Notably, ratios for both methods against IRR-MC are approximately 0.91 for *humdyst*, and 0.92 for *humprtb* and *humhdab*, highlighting the similarity in encoding size between them and the improvement both methods offer over IRR-MC.

In terms of computational complexity, the difference in number of MGPs is great; in every case, $< 6\%$ of ZZ's graph parses are required to produce a grammar of comparable compactness by the presented method. Of course, this is due to the substitution of the MGP step for each candidate constituent with the bitmask-based approximation of their gain. Although the latter is a far less complex operation, we may for a moment consider the effect if this were not true. For *humdyst*, 359,912 MGPs are required to produce a grammar of comparable compactness to ZZ, and 17,813,688 bitmasks are computed during the process. The resulting grammar contains 403 constituents, from a pool of

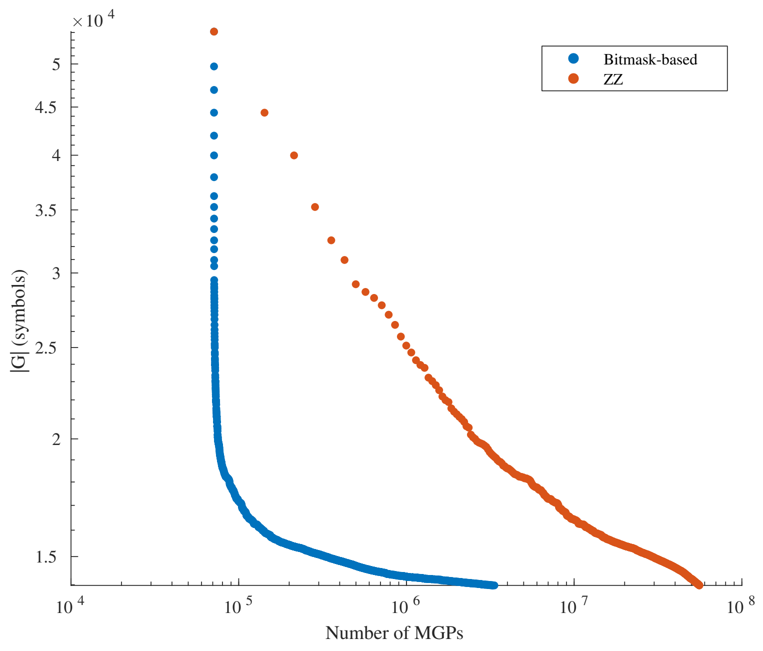


Figure 9.3: Number of MGPs to encoding length observed during construction, for *humhdab* from the DNA corpus, shown for the bitmask-based method and standard ZZ. Both axes are plotted logarithmically to highlight the difference in performance.

29,986 candidates. To generate a similar grammar, ZZ would require approximately $403 \cdot 29{,}986 = 12{,}084{,}358$ MGPs. The number of total MGPs and bitmask computations for the new method combined is $359{,}912 + 17{,}813{,}688 = 18{,}173{,}600$, which is approximately 1.5 times greater than the number of MGPs required by ZZ. Thus, it is important to examine the complexity of a graph parse in proportion to that of a bitmask operation, and the run-times of these steps in practice, to decide whether the method presented here offers better performance for this class of input. Importantly, this highlights that the scheme does not reduce the number of overall operations, but instead trades an increase in the total number of operations required for the opportunity to replace the majority of expensive operations with others which are far cheaper to compute.

An examination of the average operation times demonstrates the performance increase this strategy offers. For *humdyst*, under these experimental conditions, a single MGP requires approximately 0.33s, but a single bitmask operation requires 1.57×10^{-4} s. Operation time is measured from when the function implementing the operation is called to the time its result is fully computed. An "MGP operation" is defined as a parse of the graph for S and all constituents currently chosen for inclusion in the grammar, given a bit vector representing that combination of constituents, and resulting in a total parse cost in symbols and a bit vector of all constituents whose edges have been actively used. A "bitmask operation" is defined as an approximation of gain for a candidate constituent, given a bitmask representing the currently used nodes within the current parse graph, a bitmask representing the nodes which may be occupied by the candidate constituent, and a maximum possible gain value for that candidate. The operation results in an estimation of the total possible gain if the candidate were added to the current combination of constituents.

In the case of *humdyst*, a bitmask operation takes approximately 4.76×10^{-4} of the average time of an MGP operation, and the proportion of total MGPs to bitmask operations is 0.02, enabling the completion of 2100 bitmask operations – an estimation of gain for 1/14 of all possible constituents – in the time required to perform a single MGP resulting in an actual gain for a single constituent. Given that an equivalent, idealised grammar construction using ZZ for *humdyst* would require 12,084,358 MGPs at an average of 0.33s each, construction time using the bitmask-based method is over 35.4 times faster than ZZ.

The scheme relies on the fact that potential gains must be computed for all candidate constituents before a locally optimal constituent may be added to the current set, and these gains may be quickly computed with sufficient accuracy to prevent many MGPs from being performed before a candidate is found which produces an actual gain in encoding size. Because the approximation is sufficiently accurate in practice and does not rely on any particular attribute of a constituent, the method may be applied effectively to all input types and is capable of producing a result comparable to that of ZZ.

It may be seen that some models produced by the presented method are in fact more compact than those produced by ZZ. This is not, however, an indication of better compression, as discussed earlier; because encoding sizes for both methods are comparable in ratio, they may be considered approximately equivalent, and occur because a different *locally optimal* node in the lattice has been reached, and not necessarily because a *globally optimal path* has been discovered. This method performs occurrence optimisation in the same manner as ZZ, and is also not able to differentiate between nodes which lead to those which are are globally optimal or sub-optimal when selecting a single candidate at each iteration. Despite this, very compact grammars can be produced, and this method is shown to be as powerful as ZZ with regards to potential compression.

Canterbury Corpus

In a similar manner to the experiment conducted on the DNA corpus above, grammars were generated for three items from the Canterbury corpus; Table 9.3 contains the results, and metrics resulting from these and other associated studies. As previously described, the minimum number of MGPs required to produce an equivalent grammar using ZZ is optimally approximated by cn, where c is the number of constituents used in the encoding and n is the total number of candidate constituents during construction. Grammar sizes are reported in the number of symbols which constitute the final encoding, $|G|$.

Encoding length at each observed reduction during construction was again plotted against the number of MGPs for each piece, to show the construction rate achieved, as may be seen in Figures 9.4 – 9.6. The rate at which the equivalent grammar is constructed by ZZ is also plotted on each figure, to provide a direct indication of the improvement possible through the use of the new method.

9.2.5 Analysis

The new method was able to generate a smaller encoding than IRR-MC for two of the tested inputs only, and no models were smaller than their ZZ counterparts on this occasion. Encoding size ratios from the new method to ZZ for *grammar.lsp*, *xargs.1* and *fields.c* were 1.012, 1.008 and 1.012 respectively. Ratios from the new method to IRR-MC were 1.006, 0.991 and 0.981, highlighting the new method's failure to generate a smaller model for *grammar.lsp*. The reason for this failure is not clear from the results, and requires further investigation. However, it is not impossible for substring replacement to be performed by an IRR algorithm in a manner which produces a relatively compact model, providing the order in which replacements are made is such that only substitutions which may be retained during a ZZ-based construction are considered. Encoding size ratios from IRR-MC to ZZ are noticably lower

Sequence:	grammar.lsp	xargs.1	fields.c
$\lvert G \rvert$	1,482	1,988	3,350
Number of constituents chosen	138 of 12,717	194 of 7,439	342 of 56,044
Number of MGPs	152,662	133,757	1,227,949
Time per MGP (s)	0.078	0.081168	0.629288
Number of Bitmasks	2,804,250	1,749,421	33,031,002
Time per Bitmask (s)	6.5×10^{-5}	7.7×10^{-5}	1.21×10^{-4}
Total time (s)	12,200	11,093	779,139
$\lvert G_{ZZ} \rvert$	1,465	1,972	3,311
Number of MGPs (min)	1,754,946	1,443,166	19,167,048
Total time (min, s)	137,200	117,139	12,061,596
$\lvert G_{IRR-MC} \rvert$	1,473	2,006	3,416
MGP ratio (this vs. ZZ)	$\underline{0.087}$	$\underline{0.093}$	$\underline{0.064}$
Time ratio (this vs. ZZ)	$\underline{0.089}$	$\underline{0.095}$	$\underline{0.065}$

Table 9.3: The result of generating grammars encoding items from the Canterbury corpus using the method described in the previous chapter. Encoding sizes for ZZ- and IRR-MC-derived models are taken from studies by Carrascosa et al.

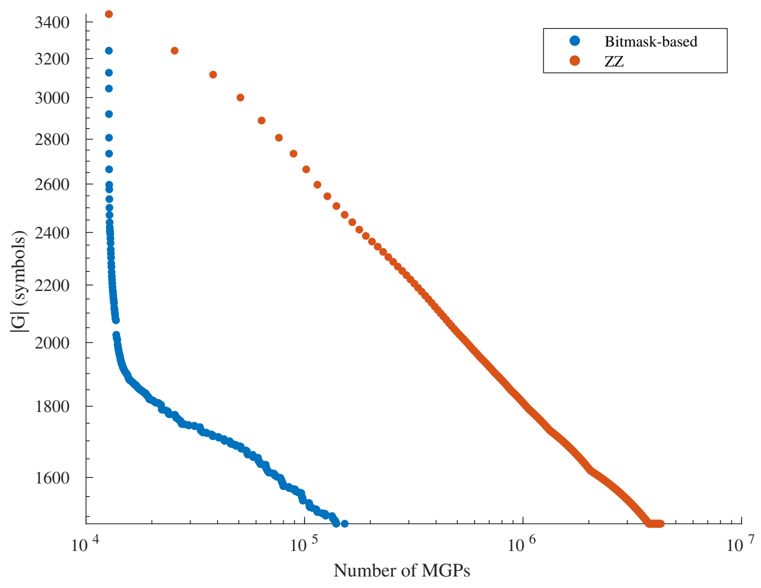


Figure 9.4: Number of MGPs to encoding length observed during construction, for *grammar.lsp* from the Canterbury corpus, shown for the bitmask-based method and standard ZZ. Both axes are plotted logarithmically to highlight the difference in performance.

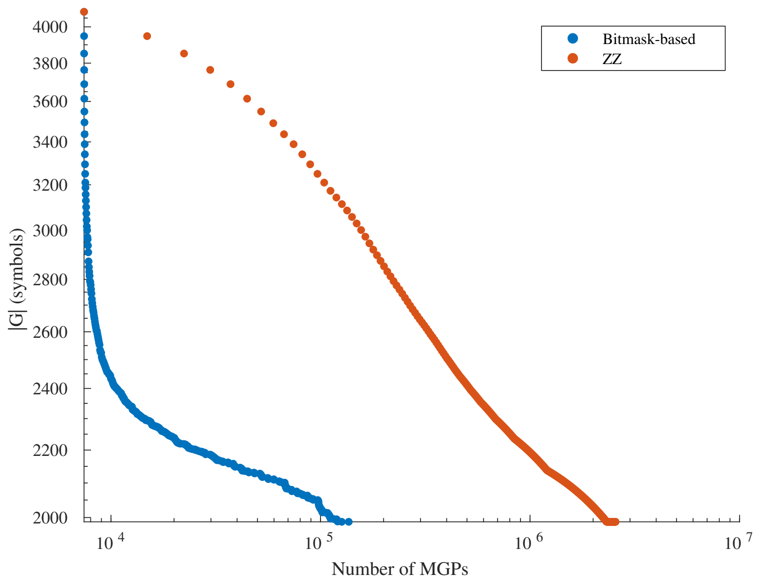


Figure 9.5: Number of MGPs to encoding length observed during construction, for *xargs.1* from the Canterbury corpus, shown for the bitmask-based method and standard ZZ. Both axes are plotted logarithmically to highlight the difference in performance.

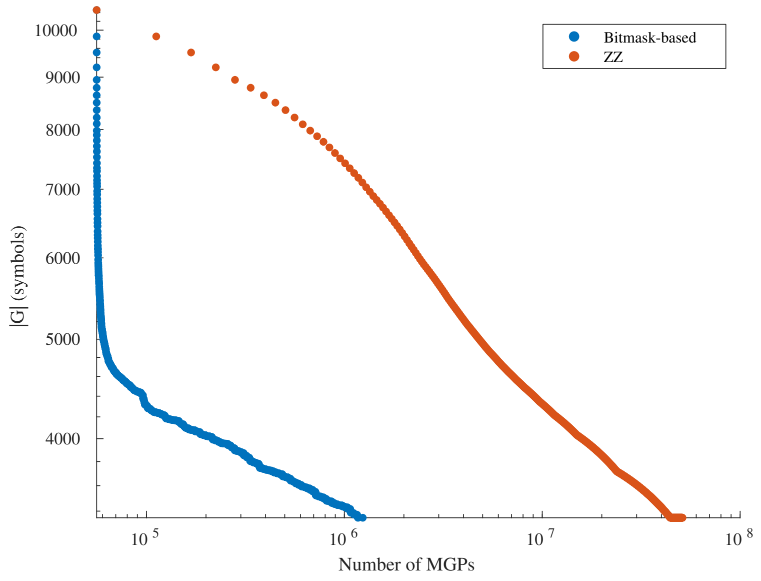


Figure 9.6: Number of MGPs to encoding length observed during construction, for *fields.c* from the Canterbury corpus, shown for the bitmask-based method and standard ZZ. Both axes are plotted logarithmically to highlight the difference in performance.

for these inputs from the Canterbury Corpus than for those of the DNA corpus, at 1.006, 1.009 and 1.032. It is possible that the chosen Canterbury Corpus inputs are relatively suitable for compression with IRR-MC, as performed by Carrascosa et al. (Carrascosa et al., 2011), or that a specific failure mode is being activated in the new method for *grammar.lsp*. Such a mode might occur because the order of constituent selection is preventing a more optimal path through the lattice to be followed, or because approximation of constituent gain is incorrectly functioning during the grammar's construction due to some particular characteristic of the input's parse graph. Isolating the cause of this failure could provide a greater understanding of the properties of the bitmask-based approach in practice, and may lead to development of an adapted approach which is able to better IRR-MC on all inputs as intended.

Once again, the difference in number of MGPs is great, with $< 10\%$ of ZZ's graph parses being required by the new method for grammar construction, for every input tested, and $< 7\%$ for the largest. Based upon the average time taken for each MGP operation, construction time is reduced to $< 10\%$ of the time which would be needed to produce a grammar with an equivalent number of constituents using ZZ in the best case, as previously described. Although the reduction in complexity seen over ZZ is less than that observed for inputs from the DNA corpus, it is not due to an increase in input complexity, as might be assumed – the inputs for this experiment are notably smaller in size and contain fewer repeating substrings. Instead, it is likely due a less uniform occurrence of substrings, as suggested by Figures 9.4 – 9.6. Unlike the curves produced during construction of models for the DNA corpus, these figures show a shallower transition from high- to low-gain for each additional constituent, and instability in the last third of the progression towards the final encoding. This may highlight a failure in the approximation of constituent gain, since each chosen candidate c_i should ideally reduce $|G|$ less than c_{i-1}.

Unlike the models generated by the new method for inputs from the DNA corpus, no models produced from the Canterbury Corpus are more compact than those created using ZZ. As discussed during analysis of the previous experiment, a single more compact encoding is not necessarily an indication of an algorithm's general ability to better compress its inputs. It is possible that the grammars output by the new method are approximately equivalent to their ZZ counterparts, as *locally optimal* constituent combinations based upon the specific configuration of the lattice seen by the algorithm during construction.

However, these limited results do not by themselves support the assertion that the new method is consistently as powerful as ZZ with regards to potential compression, and the larger encoding generated for *grammar.lsp* is a significant indication that a failure condition exists which can degrade performance to below that of the naïve IRR-MC. The number of inputs in this experiment is extremely limited, and certainly not representative of the universe of linguistic inputs. Therefore, further testing is required to assess the general ability of the method to compress, for which a more representative sample is needed. The following experiment attempts to address this with respect to the musical domain by applying the method to a large pool of symbolic inputs.

9.3 Experiment 2: Grammars constructed from the Corpus of Musical Scores

This experiment demonstrates the grammar construction method's response to musical data, and specifically to pieces from the corpus of musical scores introduced in Section 3.6, from which all input data is sourced. The experiment is conducted on pieces in order of increasing approximate complexity, as shown in Chapter 3, Figure 3.1. As described previously, complexity is approximated using the run-times required to produce normal ZZ-generated grammars for each piece. As has also been previously highlighted, grammars are computationally expensive to construct, potentially resulting in prohibitive run-times. Processing pieces in the order dictated by known construction times using ZZ allows for coverage of as much of the corpus as possible in the available time.

9.3.1 Purpose

The experiment is designed to allow a direct comparison to be made between the new method and ZZ in terms of output encoding size, construction time and complexity, in order to evaluate the level of improvement which the new method of traversal provides. It would be possible to substitute an alternative to MGP at each step which is, itself, no more efficient or effective in comparison to MGP, and yet still reduces the number of MGP operations. This experiment enables a close examination of the cost or number of operations, MGP or otherwise, which are required during construction, so that it is possible to evaluate whether the new method provides an advantage or simply an alternative to ZZ in practical terms.

9.3.2 Method

For each piece within the corpus, grammars are constructed from chromatic pitch values using the method detailed in Chapter 8, and ZZ. Rule modification is not enabled, producing grammars which are directly comparable from both methods. All grammars are fully expanded and compared to their input strings, to ensure they are able to exactly reproduce it, and thus are valid models for each data sequence. The length of the resulting encodings are stored, along with the time required for their construction. For the new method, the total number of MGP and bitmask operations are stored, along with the total time taken performing MGP and bitmask manipulation steps.

The number of MGPs required for construction of a specific grammar by ZZ may be measured empirically, although it is important to note that the grammar produced may be significantly different from that constructed via the new method, particularly in structure. There is also likely to be a difference in encoding size, given the large number of locally-optimal models it is possible to produce. These differences may make it impossible to directly compare the efficiency of both algorithms, since the results obtained are not equal. However, there is merit in attempting such a comparison; if it may be shown that one algorithm generates a smaller model in fewer operations on a given input, that algorithm is provably more efficient in that specific case. In order to capture as much of the characteristics of both algorithms in this experiment, ZZ-optimised model sizes and MGP counts will be obtained both by actual grammar construction, and by calculation of the minimum possible MGP steps required to arrive at a ZZ-built grammar with the same encoding as that produced by the new method. Calculation of this "best case" MGP count will be achieved in the manner described in Section 9.2.3 above, using the number of possible and chosen constituents in the encoding, and assuming the final node in the lattice is arrived at in the minimum number of steps.

9.3.3 Results

Of the 7928 pieces in the corpus, it was possible to process 7448 in the time available, thus covering over 93% of the available data. For 71 inputs, ranging in size from 4 to 87 symbols in length, addition of any tested combination of constituents did not produce an encoding at least as compact as the original input string. For these pieces, a construction time of 0s was recorded, along with an encoding size $|G| = l + 1$ where l is the length of the input in symbols. Such pieces were excluded from the construction time comparisons.

Figure 9.7 shows the measured grammar construction times for ZZ and the new method introduced in this study, here referred to as the *Bitmask* method. Times for each method are independently sorted, so as to present their general response characteristic. Construction times ranged from 0.358ms to 37m 17s for ZZ, and from 0.334ms to 5m 39s for the new method. Overall, the Bitmask-based method offered an improved construction time for over 96.7% of inputs, and a slower construction time for over 3% of inputs. Mean input lengths for the slower group were small, at 55 symbols out of an average length of 300.7 for the pieces tested, but with a standard deviation of 37.77. Input lengths in this group ranged from 6 to 230, and of the 234 pieces it contained, 59 could not be compressed by either method.

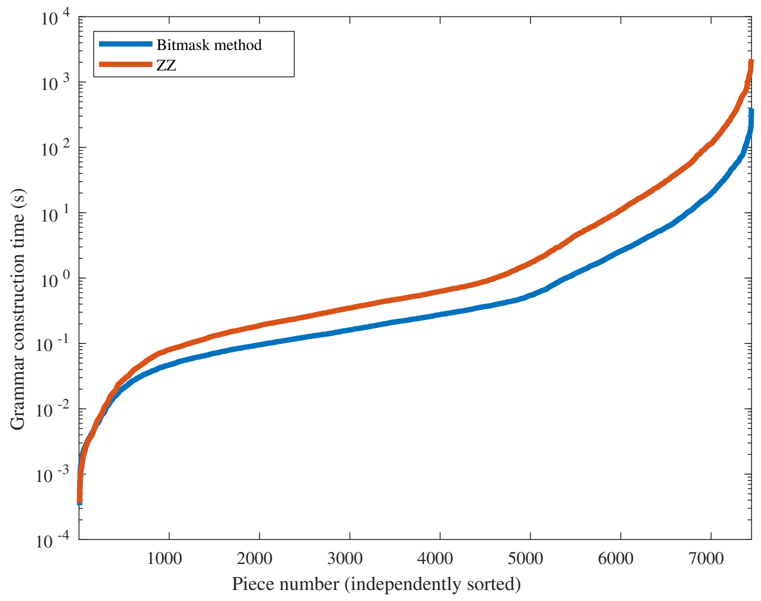


Figure 9.7: Independently-sorted construction times for grammars built using ZZ, and the Bitmask-based method introduced in this study. The y-axis is logarithmic to highlight the relationship between methods.

Average construction times across the corpus were 32.6s for ZZ, and 5s for the Bitmask-based method. Standard deviation for both groups is obviously high, at 141.5 and 19.4 respectively). The reduction in construction time resulting from the use of the new method over ZZ ranged from a -1.05s increase to a 2025.4s decrease, with an average decrease of 26.7s and standard deviation of 123.4s.

Figure 9.8 shows the range of compression achieved by both methods on the tested pieces, as a ratio of compressed:uncompressed model size. Where compression was impossible for smaller inputs, ratios range between 1 and 1.25, the latter for an input of size 4 with 1 additional terminator symbol. Graph traces appear approximately equal, and this is generally true, with over 52.8% of models exhibiting an equal size for both methods. Where unequal, there is a bias towards the new method producing larger models: almost 38% are larger, whilst the remaining 9.2% are more compact. Over all inputs, models constructed by the new method were at most 9 symbols smaller than their ZZ-produced counterparts, and at worst 58 symbols larger. However, the average size difference was an increase of 0.924 symbols, with a standard deviation of 2.51.

Overall, use of the new construction method gave an average degradation in model size of $< 1\%$ of the original input size, with a standard deviation of 0.0085. In the worst case, a model produced by the new method was $< 11\%$ of the original input size greater in encoding length than its ZZ counterpart.

The difference in model sizes between methods is highlighted in Figure 9.9. Size difference is

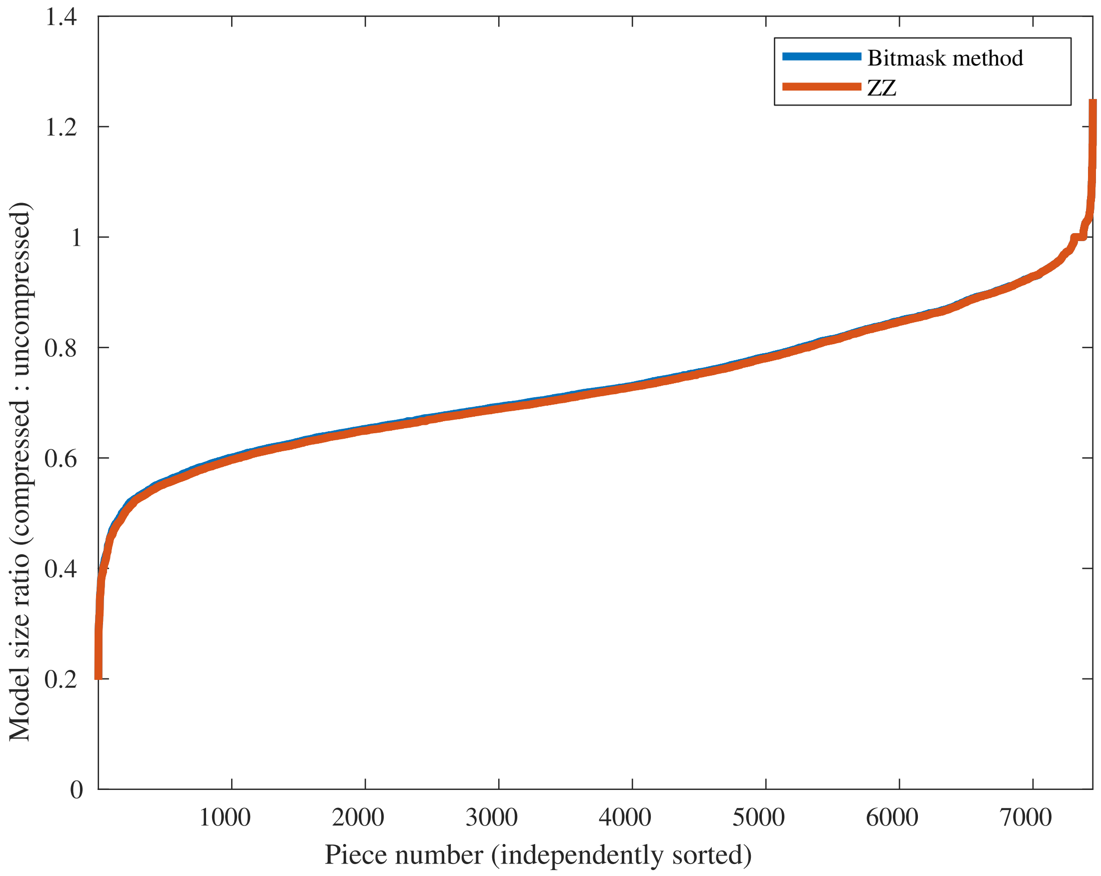


Figure 9.8: Independently-sorted ratios of compressed:uncompressed model size for grammars built using ZZ, and the Bitmask-based method introduced in this study. Model sizes between methods are approximately equal, but with a greater bias towards larger models over smaller models for the new method. Difference is ratios is too close to allow both traces to be clearly distinguished.

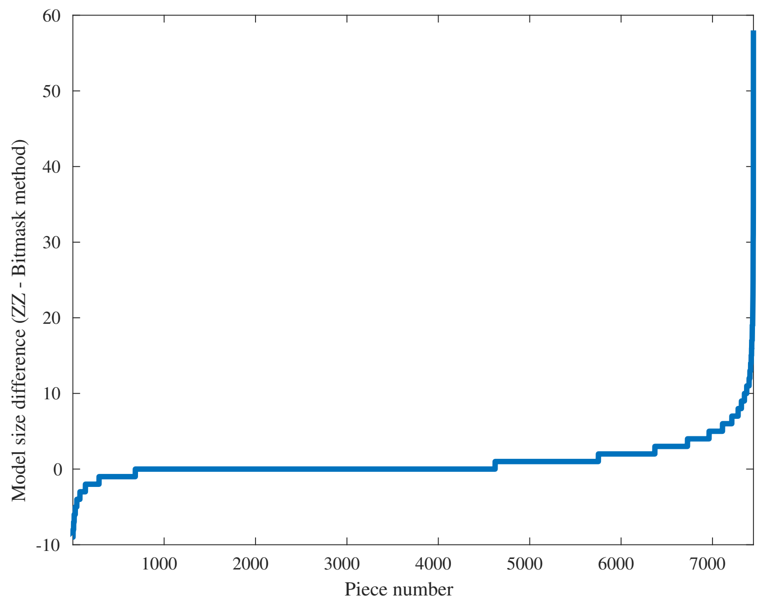


Figure 9.9: Difference in model sizes between methods, calculated as $b - z$ where sizes are b for the Bitmask-based method, and z for ZZ.

calculated by subtracting, piecewise, the size of each model produced by ZZ from the size of each model produced by the new method. Since model sizes are from the set of integers, there are many differences of equal value, giving the figure a step-like appearance. As noted above, the majority of models are equal in size, but a greater proportion of the remainder are larger.

Figure 9.10 records the number of MGPs required by both methods for construction of their grammars, and also for an estimated "best case" ZZ-produced model of equivalent encoding to that produced by the new method (i.e. one containing an equal number of rules). MGP counts are independently sorted, so that the general characteristic of each method may be observed. There is a relationship between the exponential nature of the construction times required by the corpus, and the number of MGPs performed during grammar construction. Due to this, the y-axis of the figure is logarithmic, highlighting the close relationship between response curves.

Estimated "best case" MGP counts closely track those measured during grammar construction by ZZ. In the most extreme cases, estimated MGP count is 39.7% less, and over 99% more than the measured count. However, average difference between these values is under 3.4%, with a standard deviation of 0.095. In both extreme cases, the input contained a large number of candidate constituents: 29,940 in the former case (the maximum of all tested inputs), and 7,127 in the latter (over 11 times greater than the average number of candidates over all inputs). In the first instance, ZZ used fewer constituents in practice to produce a smaller model than that generated by the new method (41:51, with model sizes 552:553 for an input of length 1109 symbols), and in the second instance ZZ used a greater number, again producing a smaller model (15:12, with model sizes 422:452 for an input of length 858 symbols).

To examine the distribution of difference in MGP counts between methods, Figure 9.11 plots the ratio of counts between the Bitmask-based method and ZZ against the length of each input. A ratio of 1 here indicates that an equal number of MGPs were required by both methods when generating a grammar from a given input, and a ratio greater than 1 indicates that ZZ was able to produce a model in fewer MGPs than the new method. The relationship between this ratio and the length of the input is highlighted by the wide spread of ratios for small inputs, and the logarithmic tailing effect as input sizes grow strongly.

The average ratio of MGP counts between the new method and ZZ is 24.1%, with a standard deviation of 0.265, showing the mean reduction in number of MGPs offered by the new method is approximately 76%. In the best case, less than 3.4% of the MGPs required by ZZ were necessary for an input of length 793, to produce a model with an encoding length of 505 symbols compared to a model of length 485 symbols produced by ZZ. That specific input contained 5,031 constituents, over 8 times greater than the average number of constituents over all inputs. In the worst case, almost 300% of the MGPs required by ZZ were necessary. However, this was from an input of length 87 symbols, for which neither method was able to produce a more compact encoding – the minimum was $l + 1 = 88$.

Across all inputs tested, the Bitmask-based method of grammar construction offers a reduction in the number of MGPs in over 99% of cases. Degradation occurred in under 1% of cases, and these involved only small inputs of < 88 symbols in length (with an average of 33 symbols, standard deviation < 18.4). An equal number of MGPs were also needed in under 1% of cases, for inputs of length < 20 symbols (with an average of 11 symbols, standard deviation < 5.87).

9.3.4 Analysis

Approximation of constituent gain during grammar construction was able to provide a significantly improved construction time for the vast majority of the tested inputs. This was achieved through minimisation of the number of MGP operations required, and replacement of the remaining evaluations with a bitmask operation which is vastly simpler and therefore faster to compute. For 175 pieces from the selection under test, although it was possible to produce a compressed encoding, construction times were not improved through the use of bitmasks, despite fewer or equal numbers of MGPs being used. Instead, the degradation is due to the additional overhead introduced by the computation

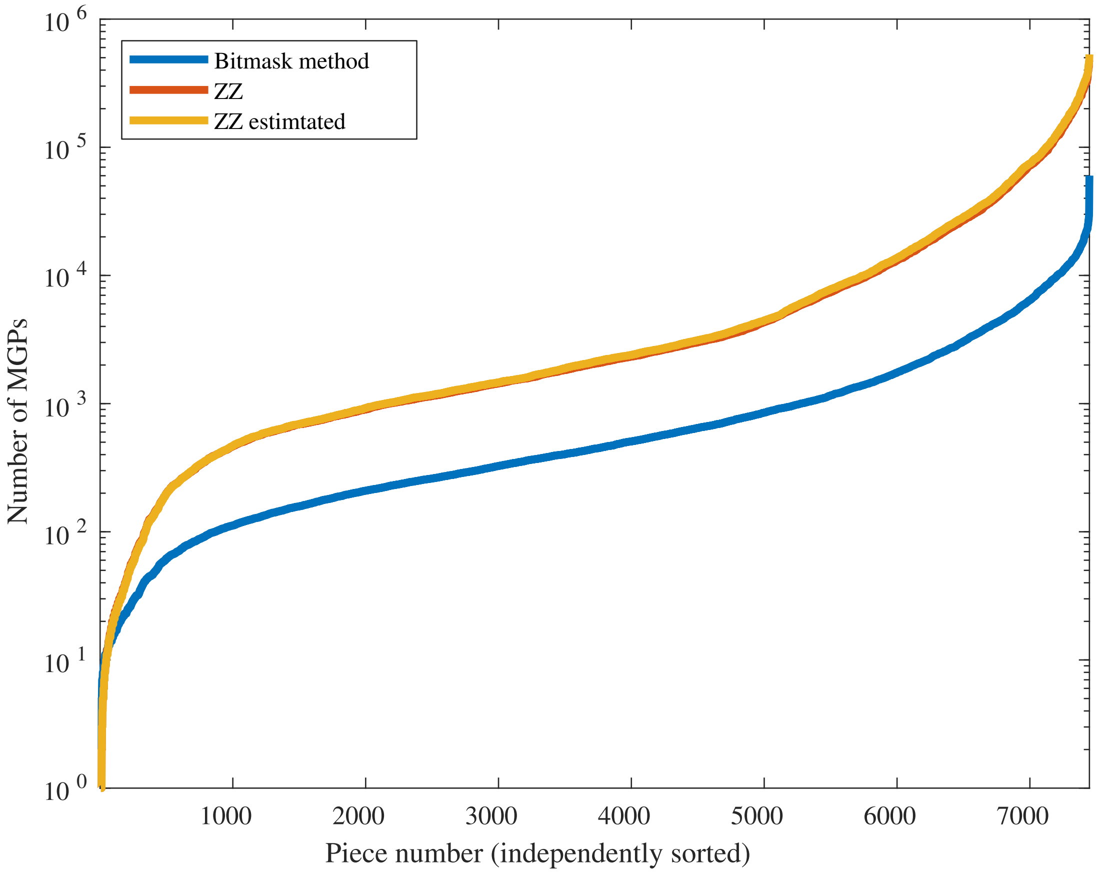


Figure 9.10: Independently-sorted MGP counts for the new Bitmask-based method (blue), ZZ (red), and a calculated best-case ZZ model equivalent to that of the new method (yellow). The estimation is so close to the actual counts that it is impossible to separately distinguish their traces. The y-axis is logarithmic to highlight the relationship between methods.

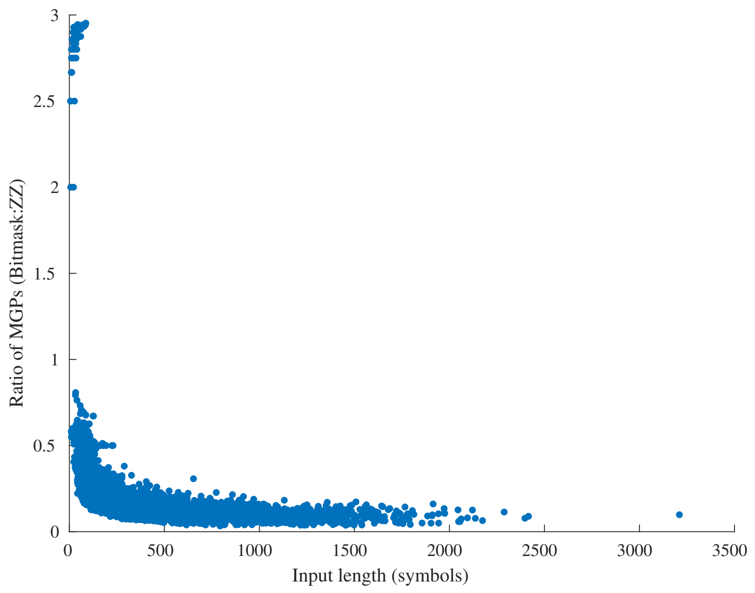


Figure 9.11: Distribution of difference in MGP counts between the Bitmask-based method and ZZ, relative to the length of each input (x-axis). A high ratio indicates the former method required far greater MGPs when generating a grammar for that input.

of the masks, relative to the very short runtimes required to construct the grammars: the maximum time within this group is 5.3s and 6.4s, for ZZ and the bitmask method respectively. It is possible to construct these grammars quickly because of the low number of constituents chosen for inclusion – on average, models in this group were composed of 1.5 constituents, with a maximum of 7 used. Since the bitmask method in this configuration requires prior knowledge of the maximum possible compression each constituent can provide when applied alone to form a grammar, construction does not begin until n MGPs have been performed, where n is the total number of candidate constituents available. In contrast, ZZ begins its traversal immediately, and is able to select its first constituent after n parses. Of these 175 pieces, ZZ produced smaller encodings for 6 inputs, and the new method produced smaller encodings for 13 inputs. Of all pieces tested, 522 were of less or equal length to the average of the group under discussion.

These observations show that the bitmask method is not a better choice than ZZ for small inputs, but it is capable of producing a smaller encoding for them in a minority of cases. Inputs of length $\leq$ 61 symbols and producing models with $\leq$ 7 production rules might be approximately considered too short for the scheme to offer any benefit in construction time over ZZ. However, for all other inputs, the bitmask method is proved capable of offering a reduction as great as $> 99\%$ of the construction time (for a small input of length 10 symbols), and 53.8% on average for the experimental data (approximately, for inputs typically ranging from 100 to 200 symbols in length). Even for large inputs, a significant reduction is possible: for one input of length 1605 symbols, the new method produced a grammar $< 0.4\%$ larger in $< 7\%$ of the time required by ZZ – a reduction of over 93%. From all pieces of length $\geq$ 1000 symbols, a minimum reduction in construction time of $> 66\%$ was observed along with an average of $> 84\%$, suggesting the method is most suitable for large inputs.

However, this increase in speed is available at the risk of marginally greater encoding lengths; 37.96% of all models generated by the new method were larger than their ZZ counterparts, by an average of 0.9236 and a maximum of 9 symbols. In the worst cases, this increase represented between 1.01% and 3.24% of the size of the ZZ-produced model, and occurred on inputs of approximately 1000 symbols in length. Given that standard deviation for the average increase in model size is 2.5074, and standard deviation of input sizes is > 329, from the results it can be suggested that an increase in encoding length of approximately 0.8% may be expected in the worst case, for inputs with the same characteristics as those tested; in paticular, this may be considered a good estimation for sequences of musical notes. The majority of the models produced by the bitmask method were equal in size to those produced by ZZ, suggesting that, although the final node reached in the lattice may be different – resulting in a different encoded structure – the outputs generated are likely to be equivalent in terms of encoding length. Processing of the results confirms that a notable number of final nodes differ between both methods: $< 31.4\%$ of all encodings were found to contain the same constituents, and $< 31.4\%$ of those which contained a different combination of constituents were equal in encoding length, showing that the new method was often able to generate an equally small model even when the same node chosen by ZZ could not be reached during traversal.

The results exhibit further evidence that the difference in encoding lengths between methods is generally very minor. As may be seen from Figure 9.8, when lists of compression ratios achieved by both methods are sorted and compared, they appear visually close to identical, especially as less compression becomes possible. A mathematical comparison of these lists reveals an average difference of 0.0029x the length of the input, with a standard deviation of 0.0014, proving that ratios achieved over many inputs are, for this data, highly similar. It is reasonable to suggest this may be a general trait as the number of inputs increases, although confirmation of this hypothesis is beyond the scope of this study. If the sizes of the generated models belong to the same *class* between methods – that is, if encodings differ because they are determined by the arrival of the algorithms at different, locally-optimal nodes in the lattice, but the paths for each include reasonable occurrence optimisation – then they would indeed exhibit individual differences in encoding length, but over the universe of all possible inputs the sum of all differences would by asymptotic. The bias to a slight increase in model size observed in this experiment may be an indication that constituent selection via the bitmask method is marginally less optimal than ZZ with respect to encoding length, or indeed that the sample size is too small to clearly demonstrate the asymptote. The fact that a minority of grammars produced by the bitmask method have smaller encodings than their ZZ counterparts offers support to the idea that a *class* of local "optimally compact" encodings exists, and both methods are capable of generating models belonging to this class.

Some models produced by the bitmask method were notably larger than those produced by ZZ. Of all models, 722 were larger than the average size difference plus the deviation, representing almost 9.7% of the encodings generated by the new method. These also represent a range of compression ratios, from 0.29x to 0.86x the size of the original input. In the worst case – an encoding 58 symbols greater than that of ZZ, for an input of length 541 – a compression ratio of 0.46, at the highest end of the spectrum, is exhibited by ZZ. The number of constituents chosen for the grammar by ZZ is greater, at 16 versus 13. Given that such strong compression is achieved, it is likely that this input possesses several conflicting candidate constituents with high potential for compression (i.e. they represent many substring instances, and are long in length), and a globally sub-optimal choice at an early stage of lattice traversal by the new method could account for the relatively large encoding size increase. In itself, this represents a known deficiency when selecting a local optimum. Although this case cannot account for all results within this group of 722 models, general compression ratios achieved by the new method once again track those of ZZ relatively closely, with an absolute maximum difference of 0.0284 between the lists of sorted ratios, again suggesting the global differences in encoding size might be relatively minor over a larger input sample. Without examination of results over a larger sample, it is impossible to concretely state which models represent outliers, or which are members of an "optimally compact" class.

Aside from a small number of failure cases for very small inputs, the results clearly demonstrate the new method offers a significant decrease in grammar construction time, and that this is related directly to its ability to reduce the number of MGPs required during construction, as shown in Figures 9.10 and 9.7. This reduction tends to steadily *increase* as input lengths grow, as shown in Figure 9.11, although the relationship is not directly proportional (i.e. the saving in construction time offered by the new method cannot match the exponential increase in complexity relative to the length of the input). This shows that although the new method is significantly more efficient than ZZ, it naturally cannot allow Occurrence-Optimised grammar construction algorithms to approach the linear complexity offered by IRR algorithms.

Although, as noted above, there is a difference between the number of observed MGPs used by ZZ, and the estimated "best-case" MGPs which would be required by ZZ to produce a grammar with the same constituents as those chosen by the bitmask method for each piece, this difference is insignificant to the purpose of the experiment. As Figure 9.10 shows, correlation between actual and estimated number of MGPs for ZZ is high, and unimportant when evaluating the improvement the bitmask method offers over ZZ in terms of MGP count and, therefore, construction time. These results also show that estimation of MGP count is a reasonable approximation of ZZ build complexity, and may be used in a direct comparison against the new method.

Examination of the ratio of MGP counts between both methods in relation to input length shows clearly that the new method does not always require a lower number of MGPs for small inputs, particularly those less than 100 symbols in length, and the number of required MGPs reduces until input lengths are ≥ 250 approximately, at which point ≤ 0.5 times the number of parses are needed. This ratio is directly related to grammar construction time, although construction and maintenance of the bitmasks introduces some additional overhead. Nonetheless, MGP count ratios may be considered a reasonable indication of best-case expected build times. Figure 9.11 shows a steady reduction in the number of MGPs required by the new method as input size increases, although the sample size for inputs of length aproximately ≥ 1800 is far smaller than those of length < 1800. The figure suggests there is a continuing trend towards a logarithmic reduction in MGP count as input length grows, indicating that the bitmask method may be highly beneficial in a situation where it is desirable to build grammars of comparable size to those produced by ZZ, from large inputs.

For the best case, where the new method was able to construct a grammar using less than 3.4% of the MGPs required by ZZ, it is important to note that the resulting encoding was larger than that generated by ZZ, at a ratio of 505:485 for an input of 793 symbols – an increase in size of over 2.5%. However, the results contain a great number of models with similar compression ratios were generated for other inputs, and only a small proportion of models constructed via the new method suffered from a large increase in encoding length. Figure 9.11 shows a significant portion of the generated models were constructed using less than 0.35 times the number of MGPs, and so a correlation between fewer MGPs and significantly increased encoding lengths is not shown to exist. It is likely, therefore, that the best-case example discussed in this paragraph is an outlier in terms of its increased encoding length, and the results show that a great reduction in MGP counts is in fact a general characteristic of the new method.

Overall, this experiment shows that a bitmask-based approach to grammar construction is appropriate for inputs which are not small – at least a few hundred symbols in length – and may offer a good advantage in terms of computational complexity for very large inputs. However, there is a stronger bias towards the production of larger encodings than ZZ, but this increase is minimal, and the new method often produced equally small models for the experimental data used. Such reduction in the number of MGPs required cannot reduce the complexity of the algorithm to that of an IRR scheme, but it does allow larger inputs to be handled than ZZ given the same allowed runtime. Providing a linear-time algorithm is not mandated by the task, the bitmask method is, in most cases, more efficient than ZZ for producing compact grammars.

9.4 Experiment 3: Grammars constructed from the Corpus of Musical Scores, with Rule Modification

Given that it is possible to construct a grammar which includes rule modifiers using a straight-line grammar constructor and lattice-traversal optimisation method such as ZZ, it is also possible to generate grammars which include rule modifications via the method presented in Chapter 8. Since the method itself does not alter the construction process, and instead reduces the time required to add a new constituent to a grammar, it needs no alterations to work with rule modifiers beyond those which are required in order to explore the additional dimension of modifiers. The increase in complexity which this space introduces is a distinct disadvantage to the use of grammars which include rule modification, and so it is certainly desirable to use any algorithm or heuristics which reduce grammar construction time in order to make the scheme practically useful. This section explores the performance of the bitmask-based method in the production of grammars which allow for rule modification.

9.4.1 Purpose

Specifically, this experiment is designed to allow a direct comparison to be made between ZZ-based and bitmask-based grammar construction algorithms, in terms of construction time and compression ratio. This is significant because without a strong reduction in construction complexity, the addition of rule modifiers to the grammar construction process is impractical for any situation where large periods of computation per model are unacceptable. As demonstrated in Chapter 5, ZZ is a useful algorithm for several musicological tasks, needing no domain knowledge to produce some musically-relevant observations. All that is required for many such tasks is the production of a model from a symbolic input sequence, and the aim of this experiment is to demonstrate whether the bitmask method offers a great enough reduction in computational complexity to allow the practical production of models which make use of rule modifiers.

9.4.2 Method

As in Experiment 9.3, grammars are constructed from chromatic pitch values for each piece within the corpus. The construction process again uses the method detailed in Chapter 8, but includes the exploration of the rule modifier dimension as presented in Chapter 6. Following that method, a grammar is first constructed which does not allow rule modification, and then each candidate modifier is added to the grammar in turn, with any modifier generating a smaller encoding added to the set of used modifiers. New candidate modifiers are tested in this manner until no reduction in encoding size occurs, and the smallest model discovered during this ZZ-style traversal of the lattice of modifiers is output as the final encoding. All generated grammars are expanded and compared to their original inputs, to ensure they model each piece correctly. The length of the resulting encodings are stored, along with the time required for their construction.

Three specific construction methods are included in the investigation:

1. Bitmask-based, with rule modification (producing a flexible grammar)
2. ZZ-based, with rule modification (producing a flexible grammar)
3. ZZ-based, without rule modification (producing a standard grammar)

9.4.3 Results

For this experiment, it was possible to process 6100 pieces in the time available, of which 5718 were also processed in Experiment 9.3, thus covering over 72% of the available data in a manner which allows piece-wise alignment of the results. For 68 inputs, a model which with a more compact encoding than the original input could not be found when building a grammar which either did not allow rule modification, or a grammar which did allow modification but was constructed using the bitmask method. By comparison, use of the ZZ-style traversal presented in Chapter 6 resulted in a failure to produce more compact encodings for 66 inputs.

Figure 9.12 shows the measured grammar construction times for the three methods. Again, times are independently sorted, so as to present the general response characteristic of each method to the tested inputs. Construction times for method 1 (bitmask-based, with rule modification) ranged from 498ns to 11m 10s, compared a range of 34ms to 18h 54m for method 2, and 27ms to 3m 16s for method 3. Method 1 was able to generate grammars faster than method 2 in every case, and faster than method 3 in over 6.7% of cases. It was, however, slower than method 3 in over 93% of cases. Average construction

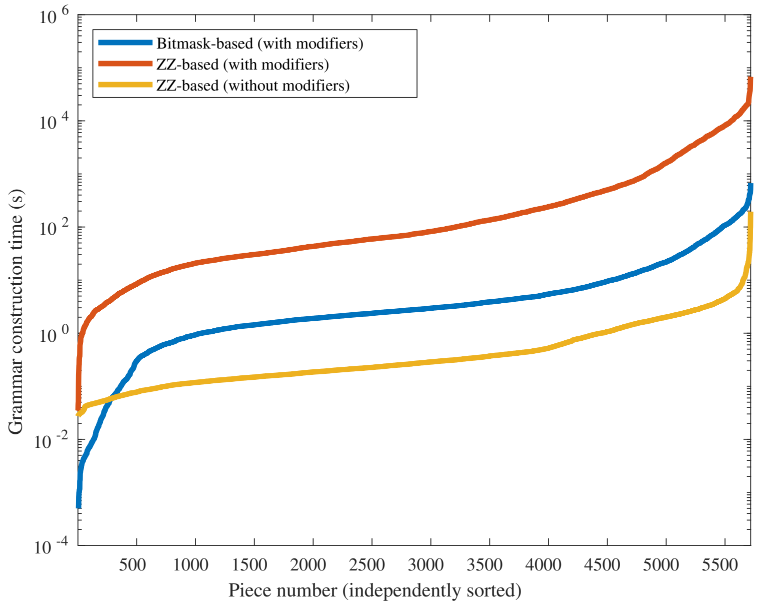


Figure 9.12: Independently-sorted construction times for bitmask-based and ZZ-based grammars which include rule modification, and ZZ-based grammars which do not include rule modification. The y-axis is logarithmic to highlight the temporal relationship between methods.

times were 15s for method 1, 18m 7s for method 2, and 1s for method 3, with preditably large standard deviations of 41.48s, 58m 41s, and 4.5s for each group respectively. Over all inputs and on average, method 1 was able to construct its grammars in under 0.044x the time required by method 2, and needed almost 13.17x the time required by method 3. Standard deviation for these averages were 0.045 and 12.896 respectively.

Figure 9.13 shows the compression ratios which each method achieved on the input data, represented as independent plots of compressed:uncompressed model size ratios. As noted in Experiment 9.3, a grammar with a smaller encoding than the original input string could not be produced for some small inputs, and these are represented by ratios ≥ 1 in the figure. Graph traces appear approximately equal, and the maximum observed differences in output encoding size were 68, 28, and 83 symbols between methods 1 & 2, 1 & 3, and 2 & 3 respectively, for which ratios in each group were 0.6826 & 0.5274, 0.6418 & 0.5582, and 0.6904 & 0.8640. In each case, these differences represented an increase of less than 16%, 9% and 18% in model size. On average, models produced by method 1 were < 3.58 symbols larger than those produced by method 2, and > 0.04 symbols smaller than those produced by method 3, with standard deviations of < 7.05 and > 3.27 respectively. Relative to the original inputs, method 1 produced models which were on average $< 1.6\%$ larger than those produced by method

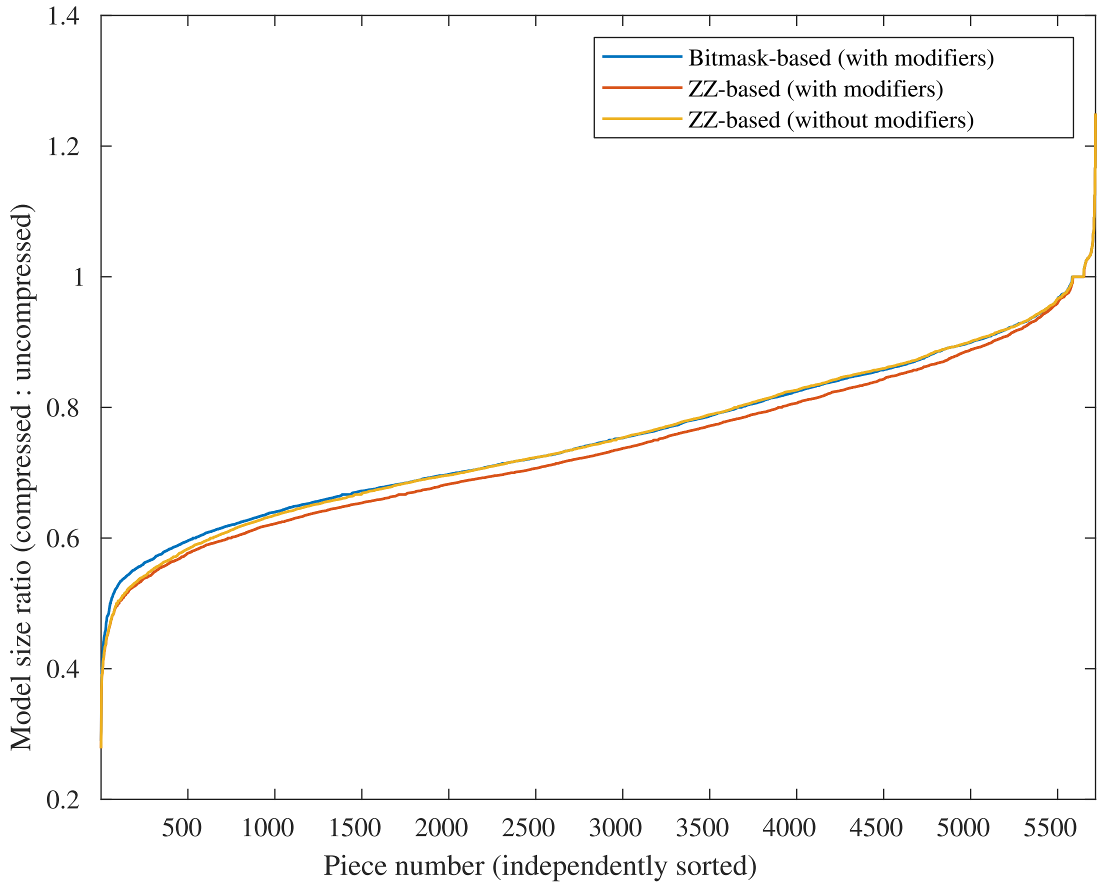


Figure 9.13: Independently-sorted ratios of compressed:uncompressed model size for grammars allowing rule modification built using bitmask- and ZZ-based methods, and for grammars without rule modification built using ZZ. Model sizes between methods are approximately equal, but with some variation between methods.

2, and $< 0.24\%$ larger than those produced by method 3, with standard deviations of < 0.025 and > 0.016.

Overall, use of the new construction method gave a degradation in model size for approximately 53.4% of all inputs, relative to method 2, and an improvement for just under 8% of inputs. Output encoding size was equal in approximately 38.6% of cases. In comparison to method 3, the new construction method gave a degradation in model size for approximately 34.9% of all inputs, and an improvement of under 8%, equivalent to that shown against method 2. Output encoding size was equal in approxiately 57.2% of cases.

Figure 9.14 charts the difference in model size between the three method groups 1 & 2, 1 & 3, and

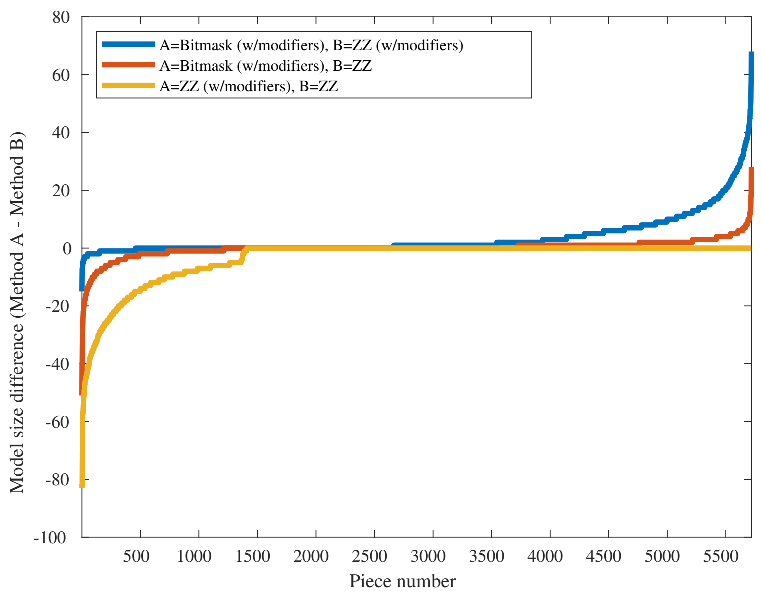


Figure 9.14: Difference in model sizes between methods, calculated as $a - b$ where sizes are a for the first method, and b for the second.

2 & 3. Grammars produced via the bitmask-based method exhibit an increase in size for some inputs, relative to those produced by method 2 or 3. However, no such increase occurs between ZZ-based methods.

9.4.4 Analysis

In every case, the use of bitmasks in approximating constituent gain during grammar construction allowed grammars to be constructed more quickly than those which relied on MGPs to achieve the same aim. On average, construction time was reduced by over 22.7x, and times required by the bitmask-based method were closer to those needed by ZZ when disallowing rule modification than to those when rule modification is allowed. In the best case, construction time was reduced by over 309x to produce a grammar with an encoded length of 481 symbols, compared to an encoding of length 442 from method 2, from an input of length 693 symbols.

In almost 8% of cases, the bitmask-based method was able to produce smaller models in a shorter time than method 2. Those models were, on average, 1.56 symbols smaller and constructed 19.8x faster, with standard deviations of < 1.21 and < 18.5 respectively – this was true for 457 models, comprising nearly 8% of all inputs.

However, in the majority of cases the sizes of the models produced using method 1 did not have encodings which where at least as small as those produced by method 2 – the latter condition occurred for 46.59% of all inputs, where average difference from method 2 was > -0.267 symbols with standard deviation 0.7717. This difference is significantly small, and over 17% of the models from this group featured a smaller encoding length. Where method 2 generated smaller models, the average difference was > 6.92 symbols with standard deviation 8.2702. These results indicate that grammars of a comparable class in terms of size my be produced using the bitmask-based method, for a large proportion of inputs of this type, but it is more likely a larger grammar will be produced. Relative to the original inputs, larger models had an encoding length which was, on average, $< 3.2\%$ greater, with standard deviation 0.0265. By comparison, models produced by ZZ for these inputs had an encoding length which, on average, increased by $> 2.5\%$, with standard deviation 0.0349, when rule modification was disallowed.

These findings demonstrate a weakness in the bitmask-based method when attempting to produce comparable grammars to method 2, with encoding lengths more strongly resembling those of models produced by method 3. However, in the majority of cases method 1 is capable of producing smaller grammars than method 3. Examination of the mean suggests the potential reduction is minor, with a mean saving of > 0.044 symbols, although deviation here is high at over 3.27 symbols – where method 1 produced smaller encodings than method 3, the average reduction was > 3.75 symbols with a deviation of > 4.7, showing that there is a clear split between cases where the new method was, and was not, able to further compress the input. Given the practical reduction in construction time the bitmask-based method achieves, it may be said to be suitable for applications where grammars which include rule modifiers must be quickly constructed, and an increase in model sizes of approximately 3% over ZZ-based grammars is tolerable.

It is reasonable to suggest that the strong decrease in computational complexity the method allows is also strongly limiting the number of nodes in the lattice of all possible constituent combinations that is being explored, which in turn is preventing the discovery of smaller local minima. Despite the fact this equates to an inability to fully rival method 2 in compression strength, the bitmask-based method is still capable of producing smaller rule-modifier models from music data than a two-dimensional ZZ in the majority of cases, and can do so in a fraction of the time.

9.5 Experiment 4: Exhaustive Discovery of Locally-optimal Nodes in the Lattice

All the experiments in this chapter have resulted in the production of models which are often measurably larger when constructed using the bitmask-based method, than when constructed using standard

ZZ, regardless of whether rule modification is enabled. The analysis in Section 9.4 tentatively suggests the possibility that a *class* of models exists, within which all encodings may be considered "sufficiently compact", and represent nodes within the lattice of candidate constituents which are locally optimal – that is, no further minimisation of $|G|$ can occur if a constituent is added or removed from the currently chosen set of constituents, represented by the inability to move to a neighbouring node in the lattice which offers a reduction of $|G|$. The space of all possible constituent combinations which the lattice represents is large, at 2^n nodes given n candidates which must be considered, and an optimisation scheme which approximates the global minimisation of $|G|$ by allowing a local optimum to be selected is able to arrive at any of the set of locally-optimal nodes present within the lattice, of which one or more is the global optimum.

How, then, might the performance of two algorithms which arrive at different nodes which are both locally optimal be compared? One possibility is to exhaustively explore the space of all nodes from which no improvement is possible, and examine the set of model sizes present within it. Where the proportion of nodes which represent the global optimum to all nodes the space contains is high, the algorithm does not need to explore the space in a particularly rigorous, intelligent, or exhaustive manner in order to discover an ideal model. Equally, where the proportion is very small – perhaps only one optimum node is present within an intractably-large space, for example – it may be impossible for an algorithm to arrive at the global optimum without beginning in a state where its actions will, cirumstantially, guide it directly to the ideal node. If it were possible to know the space of all possible solutions *a priori*, the use of an optimisation algorithm would be redundant. Instead, it may be useful to discover any generalised characteristics of the solution space, so that any pair of solutions may be considered relative to each other, based on the sizes of the models they generate given knowledge of the position which one of the solutions occupies within that space.

9.5.1 Purpose

Given that it is combinatorically difficult to construct a model which reasonably approximates the smallest grammar, this experiment aims to explore the hypothesis that there exists generally a distinct and limited set of "compact" models, and a far larger set of models which do not approximate the smallest grammar well, for any given input. It is designed to explore the space of all nodes in the lattice which minimise $|G|$, using a large population of inputs, in order to discover the distribution of model sizes within the space, so that the characteristics of that distribution might be used as an aid in the evaluation of the "compactness" of one model relative to another, as an approximation of the performance of a grammar-building algorithm. The experiment is presented here as a simple proof of concept, which may be explored more thoroughly in future work.

9.5.2 Method

Pieces from the corpus of musical scores were chosen for use as inputs in this experiment, since the corpus consists of a large number of pieces, they are all from a particular domain – in this instance, musical – and are relatively small in length compared to the DNA and Canterbury corpora, enabling the processing of as many inputs as possible in the available time. As in previous experiments, pieces are processed in order of their approximate construction complexity.

For each input, each repeating substring is located and stored, to produce a list of all possible candidate constituents. Only substrings whose instances exactly match each other are considered; no rule modification is allowed. Beginning with the set of constituents $C = \emptyset$, all possible orders by which candidates may be added to C are tested, and only those which result in a reduced $|G|$ are retained – any which result in an increase in encoding size are discarded. A history is kept of each C considered. When a combination is reached from which no addition or removal of a candidate constituent is possible without increasing $|G|$, the combination is stored as a "locally optimal" node, and traversal continues from the last C where it is possible to choose an alternative candidate which does not degrade $|G|$. Exploration of the space is terminated when all possible combinations of C have been evaluated for the given input. In this manner, every "locally optimal" node within the lattice is located and stored, along with the size of the encoding the combination generates, providing a map of model sizes present within the lattice.

Once these nodes are known, a histogram for observed model size counts is calculated for each piece. The smallest encountered model represents the global optimum encoding, and the largest model which it is possible to construct is simply the input sequence plus a termination symbol, without any production rules which reduce $|G|$. In order to allow a comparison between inputs to be made, model sizes are represented on a normalised scale, where 0.0 is the most compact encoding possible, and 1.0 is the least compact. A number of bins is chosen, into which the model size counts are placed, and finally an average is computed across all inputs. Where the difference between the smallest and largest encodings encountered is smaller than the number of bins for a given piece, its encoding space is interpolated so that a characteristic curve is maintained across all bins, in an attempt to enable a direct comparison between pieces with varying encoding length ranges without the introduction of statistical artefacts.

Observed model size counts are also separated into three groups, based upon the total count of models of any encoding length which exist for each piece. Boundaries are chosen so that each group contains a reasonably large number of pieces relative to any other group, whilst maintaining a useful and characteristic response. Model size counts are converted to proportions of the total number of models per piece, to enable the normalised response curve of each to be compared, and a general average response to be computed for the group.

9.5.3 Results

In the time available, it was possible to entirely process 4,329 inputs from the corpus, covering over 54% of all pieces in the collection. The average difference between the minimum and maximum sizes seen per piece was under 32 symbols, with a standard deviation of > 21 and a maximum of 347. In order to provide a good historgram resolution, 100 bins were chosen within which to represent the results. Figure 9.15 shows the average counts observed, along with their variance.

Figures 9.16a – 9.16c show the average proportion of locally-optimal models observed, relative to the total number of models observed per piece. To maintain a clear indication of the response, groups were manually chosen with counts of 1–20, 21–200 and 201+ models, containing 3,472, 541 and 316 pieces respectively. The number of pieces for group 1 was high due to the large number of smaller inputs processed, and, due to an exponential growth in piece length within the corpus, a naturally strong decrease in pieces per group occurred as input size increased, and it became possible to construct a greater number of models per piece.

9.5.4 Analysis

There is a distinct and relatively smooth curve to be seen in Figure 9.15, showing a steadily-growing increase in the average count of models of a particular size as model compactness increases. The average count peaks in the range 0.109-0.119, and follows a reducing curve to an average of 3 most compact models. The average count of globally optimal models was 1.9109, with a variance of 3.2066. This is distinctly different from the highest model count around the range 0.109-0.119 of 7.83, with standard deviation 22.1239, where there are approximately 4 times the number of nodes which locally minimise $|G|$. However, standard deviation is overall very large, showing great instability which grows and reduces in magnitude along the curve of the average. This unfortunately highlights the fact strongly different results may be expected between different inputs, and, as such, the presence of a large or small count of models of a given size cannot be used as a direct indication of the position of that solution within the space of model size counts. Nonetheless, on the chosen input data at least, there appears to be a definite curve, and it is potentially possible that a sample of several model size counts for a given input may be used as an indication of their general position within the solution space. If

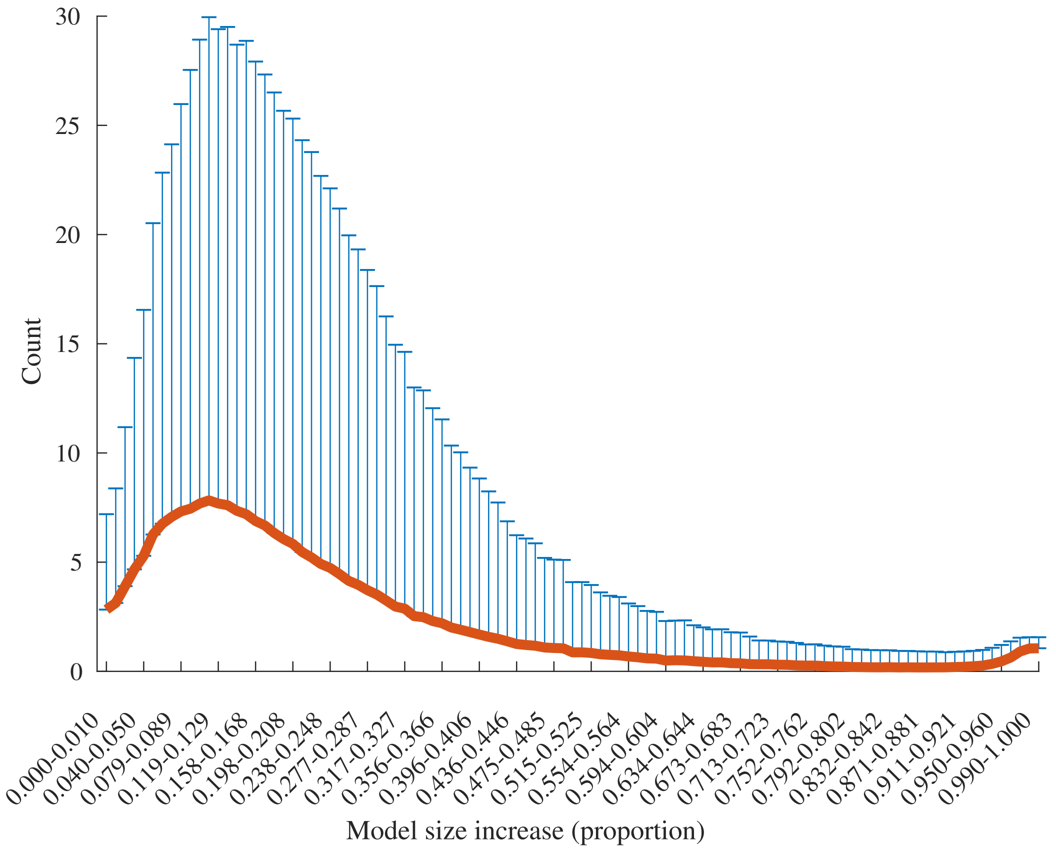
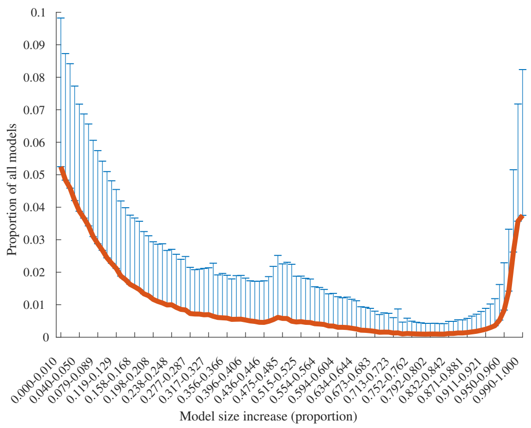
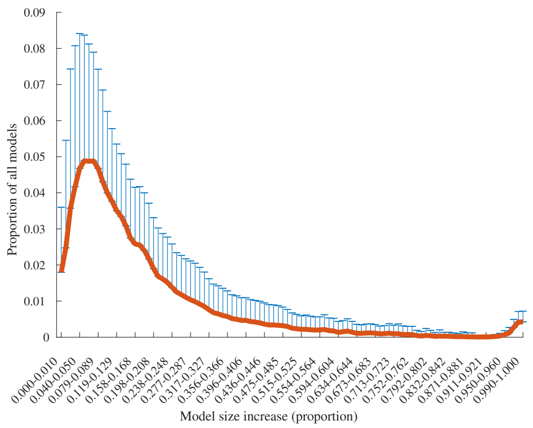


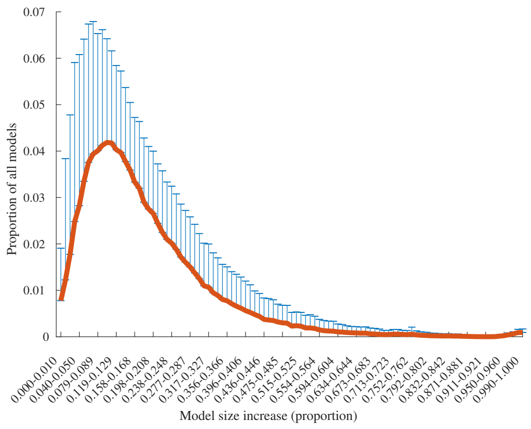
Figure 9.15: The average number of locally-optimal models existing within the lattices for 4,329 pieces from the corpus of musical scores, with standard deviation shown as error bars. Model sizes are represented as a proportional increase from the global optimum (0.0) to an uncompressed encoding (1.0).



(a) *Group 1: pieces with model count 1–20*



(b) *Group 2: pieces with model count 21–200*



(c) *Group 3: pieces with model count 201+*

Figure 9.16: The average proportion of locally-optimal models existing within the lattices, over three groups based on total model count per piece, with standard deviation shown as error bars. Model sizes are represented as a proportional increase from the global optimum (0.0) to an uncompressed encoding (1.0).

it were possible to predict an approximate position, this information might be used to guide choices made during constituent combination optimisation, perhaps as a heuristic used during construction, but the comment is included here merely as a suggestion for future work, and exploring the idea is beyond the scope of this study.

As previously described, results segmented into three groups based upon the total number of possible models per piece, and the proportion of model counts per bin used to enable response curves of differing amplitudes to be normalised, and directly compared (Figures 9.16a – 9.16c). As the number of models which may be constructed for a given input increased, a definite clustering of locally-optimal models which were significantly larger than the global optimum occured. The first group (1-20 possible models per piece) did not contain this characteristic, instead displaying a peak average of 0.0525 in the range 0.0-0.01, corresponding to an average of 3.37/64.3 models. This steady increase may be attributed to the shortness of inputs within this group: encoding length differences between various constituent combinations results in a spread of possibilities over a small number of discrete model sizes as the global optimum is approached, represented by the steady increase in model counts over a relatively large portion of the x axis. Inclusion of any useful constituent is likely to result in a significant reduction in $|G|$ given the short length of the original string, and this causes a sudden decline in model counts as $|G|$ reduces from the global maximum.

The second and third groups, representing models of increasing complexity, showed a strongly increased probability for the selection of a model which was not globally optimal, given a random choice between groups of constituent combinations which generate exactly one unique model. For the group of pieces with a total model count of 21-200, the maximum proportion of model counts was 0.0489 on average, and occured in the range 0.05-0.059. In contrast, the proportion of smallest models was 0.018, representing an average of 6.97/387.94 models. For the group of pieces with a total model count of 201+, the maximum proportion of model counts was 0.0419, occurring in the size range 0.099-0.109, and the proportion of smallest models was 0.0078 (corresponding to an average of 13.97/1795.58 models). As can be seen in the figures, variance also decreased as total model count increased between groups, suggesting the observed tendency became more stable as the space of all possible models grew in complexity. Notably, the curve became broader in Figure 9.16c, and the peak average model count occured for a proportionally larger model than all other groups. Additionally, he count of smallest models was shown to be greatly reduced. Separation and representation of the results in this way demonstrates the apparent trend shown in Figure 9.15 does not exist purely as a statistical anomaly, and was strongly associated with an increase in lattice search space complexity. This is perhaps supported by the empirical observation that moderately compact models can be generated with relative ease, but the discovery of smaller models becomes increasingly more difficult.

Despite the presence of this characteristic, the hypothesis that a distinct group of "compact" solutions exist within the sampled space cannot be upheld by these results. However, it is clear that very few highly compact solutions exist on average given any sufficiently complex input, relative to a high count of sub-optimal solutions, suggesting this is a general characteristic of the search space generated by that input. As such, it may be suggested that a class of "sufficiently compact" solutions could be defined, based on each individual application, and used to discriminate between models of a desired target range in size and those which are to be rejected. Overall, an average of approximately 231 locally-optimal models were discovered during the experiment, making the proportion of optimal and highest-count solutions present within the space 1.22% and 5.21% respectively. The response curve observed is perhaps most likely caused by two main factors: the decreasing probability of encountering a large model from which no improvement can be made from 0.115-1.0 approximately, and the decreasing probability of discovering a new constituent compatible with the current combination which reduces $|G|$ from 0.115-0.0. Discovery of the factors which actually form the response is left for future work.

For a grammar being constructed using ZZ, if groups of constituents which lead to a unique leaf node are considered in a random order, the probability that the algorithm will reach the optimal solution given the same type of input data used for this experiment is approximately 2.83/231 on average. Consequently, the probability that the algorithm will return a sub-optimal result is 228.3/231, or 98.8%. For the group of pieces with 201+ possible models shown in Figure 9.16c, these values degrade further: the probability of reaching the optimal solution becomes approximately 9.84/1796, and the likelihood of obtaining a sub-optimal result becomes 99.45%. In this manner, the expectation of achieving a particular value of "compactness" on the scale shown in Figure 9.15 can be estimated, providing the global minimum for $|G|$ is first known.

Approximate Evaluation of Algorithm Performance On the basis of these results, it is possible to approximate the performance of the new method for the earlier experiments in this chapter, but very loosely – the following comments cannot be considered definitive, as the results show that the model size count characteristic is strongly linked to the individual input, and the experiment does not provide any evidence that these results generalise to other inputs, of identical or different types. With these limitation in mind, a solution's position on the normalised model size count scale shown in Figure 9.15 may be calculated as $g - x/g - s$ where g is the length of the input in symbols + 1 and represents the largest possible encoding, s is the length of the smallest possible encoding, and x is the observed size of the model which is to be evaluated.

In Section 9.2, models were generated which were larger than those produced by ZZ for inputs *humprtb* from the DNA corpus, and *grammar.lsp* from the Canterbury Corpus. The latter, in particular, was larger than a model constructed using IRR-MC. For *humprtb*, if it is assumed that the ZZ-produced encoding of length 13,658 is approximately equal to the global minimum for $|G|$, the encoding produced by the new method exists at 2.09×10^{-4} percent on the scale of model sizes. This is very close indeed to the global optimum, and falls within the histogram bin chosen in this experiment to contain the minimum value for $|G|$. However, the encoding produced by IRR-MC exists at 0.028598. This value falls within the second histogram bin, and for the purposes of this discussion might be considered roughly "distinct" from the "most compact" models. For *grammar.lsp*, making the same assumption for the ZZ-generated encoding, the new method's model is present at 0.007532, and the model generated by IRR-MC exists at 0.003545. Both are again very close to the global optimum, and can tentatively be considered sufficiently small, or "most compact" encodings on this scale, albeit from an input of a different type to that evaluated in this experiment.

In Section 9.3, it was observed that use of the new method produced an average increase in encoding size of 0.9236 symbols for pieces from the corpus of musical scores, with a standard deviation of 2.5704. The average input length for the tested pieces was 405.8 symbols, although with a large standard deviation of 763.2 which, for now, will be ignored. The average size of models generated by the new method was 222.8 symbols, with a deviation of 239.5. Very approximately, this places the average model size at 0.005022 on the scale shown in Figure 9.15, again well within the "most compact" range of $0.0 - 0.01$ chosen for this discussion. An exploration of individual pieces, with discovery of the position the new method's models occupy within their own space of locally-optimal models, would be necessary before a generalised and accurate evaluation could be made of the "compactness" of the encodings produced. However, this initial, simple calculation does not immediately discount the suggestion that the models generated by the new method in the experiment belong to the same "compactness class" as those produced by ZZ. The results in Section 9.4 are even stronger, with an average increase in model size of < 0.268 symbols. It is still notable, however, that there is a bias towards the new method producing larger models. Further experiments would be necessary to discover whether this is in fact linked to the decreased probability of arriving at a more optimal node given a different ordering of candidate constituents, a failure mode in the approximation method used which may be addressed, or simply the inability of the algorithm to explore the lattice in a sufficiently optimal manner.

9.6 Summary

This chapter presented the investigation of a novel method of lattice traversal, which reduces the number of MGPs required during grammar construction by dynamically approximating the gain each candidate constituent might offer to the current iteration's solution, before selecting the first from a descending list of gain values which produces a reduction in the encoding length of the grammar. These investigations were designed to empirically determine whether this novel method was able to construct grammars which were equally or approximately as compact as those produced by ZZ, but in a significantly reduced run time.

In every experiment conducted, the method was able to generate grammars in a fraction of the time required by ZZ, due entirely to the replacement of the majority of the expensive Minimal Grammar Parsing operations with bitmask-based approximations which are computationally simple, and therefore faster to perform. Where inputs from the DNA corpus were processed, the number of MGPs was reduced by $\geq 94\%$. For the Canterbury Corpus, this dropped to $\geq 90\%$. Where pieces from the corpus of musical scores were compressed, this reduction varied from approximately $\geq 65\%$ for small inputs to $\geq 75\%$ for larger inputs of around 1500 symbols. Model sizes were not exactly equivalent to those produced by ZZ: for inputs from the DNA corpus, two models had a smaller encoding length than their ZZ equivalents, but in every case, the models were smaller than those produced by IRR-MC. For inputs from the Canterbury Corpus, no models were smaller than those generated by ZZ, and one model was larger than its IRR-MC equivalent. In all cases, model sizes were close to those produced by ZZ. For inputs from the corpus of musical scores, 52.8% of models were equivalent in size to those produced by ZZ, whilst 9.2% were more compact, and 38% were larger by an average of 0.924 symbols.

Where the method was employed to generate grammars which allow rule modification, it was able to do so in $< 0.044x$ the time required by a ZZ-based constructor also allowing rule modification, due to the reduction in the number of MGPs required. The sizes of models produced using the new method were equivalent to those produced by ZZ for 38.6% of inputs, whilst $< 8\%$ were more compact, and 53.4% were larger by an average of 3.58 symbols. Compared to encodings produced by ZZ with no rule modification allowed, models generated by the new method were, on average, > 0.04 symbols smaller, with reduced model sizes occurring for $< 8\%$ of all pieces, and increased model sizes for 34.9%. This discrepancy might be explained by the greatly increased search space that must be explored when producing a grammar which includes rule modifications, and represent selection of a local optimum which is not as compact as that arrived at by ZZ without the addition of rule modification.

Finally, the space of all constituent combinations whose encoding length $|G|$ cannot be reduced by adding or removing any constituent was explored for 4,329 pieces from the corpus of musical scores, and averages of the normalised counts of model sizes existing within this space calculated and plotted. This showed a large proportion of moderately-compact models exist on average, relative to a very small number of highly-compact or globally optimal models. Although in no way offering a definite approach to assessing the "compactness" of a given grammar, the experiment's results highlighted the likelihood that the increased model sizes observed throughout the chapter did not represent a failure of the bitmask-based method of constituent selection to produce models of comparable "compactness" to their ZZ-based counterparts.

A wise man once said that to do a great and important work, two things are necessary: a definite plan, and not quite enough time.

R.C. Schafer, 1977

10
Conclusions & Future Work

This chapter summarises the conclusions of all investigations throughout this thesis, and highlights possibilities for future research which suggest themselves as interesting and promising given the study's outcomes. It is divided into three sections: standard grammars in application to music data in Section 10.1, the addition of rule modification to grammars in Section 10.2, and using the approximation of constituent gain to improve the speed of lattice traversal during grammar construction in Section 10.3. In each section, the salient features of the outcomes of each experiment are listed, and placed into the context of the purpose of the investigation, along with any inference or important observations it were possible to make from the results.

Overall, it is shown that grammars can be leveraged to perform some musicological tasks, but are unable to offer performance as strong as existing methods in every case. The addition of rule modification, a simple but versatile framework which allows custom transformations to be included within a grammar's encoding, allows the construction of moderately smaller models, in particular for musical data, but at an exponential increase in computational complexity. However, replacing an expensive parsing operation with a simple bitmask-based approximation of the gain a candidate rule may offer during grammar construction allows an exponential decrease in computational complexity, in practice. This improvement in speed is a trade-off which causes slightly larger encodings to be generated, and cannot counteract the increase in construction time required by the addition of rule modifications during the creation of a grammar, but it is an aid to making the process practicable and allows the generation of standard grammars in significantly shorter periods than is currently possible using

ZZ.

10.1 Standard Grammars in Application to Music Data

Chapter 5 investigated the application of grammar-based compressors to six practical musical applications, comparing their performance to that achieved by the use of other popular compression algorithms. The responsiveness of each method to errors was examined, and their performance when applied to location of errors, classification of folk music by tune family, and discovery of expert-defined musicological patterns was measured.

When tasked with detection of a common transcription error, LZW proved most responsive for pieces too small to compress by standard grammar, but beyond this margin ZZ proved most sensitive. All methods showed a logarithmic response to an increasing number of errors, with GZIP outperforming ZZ as input length became significant. ZZ was most successful in correctly identifying the position of a single error, with an F-measure of $0.22 - 0.35$. Strong variation in response was measured for all methods, showing none can be relied upon to respond correctly in each individual case. However, every method generated a larger model in the majority of cases.

When grouping pieces from the Meertens Tune Collections by tune family, ZZ was able to perform moderately by comparison with existing studies, when the results from multiple representations were weighted and used in nearest-neighbour classification. ZZ was also applied to the discovery of expert-defined patterns from the polyphonic *Discovery of Repeated Themes & Sections* task presented in MIREX 2016, where it bettered all submitted methods for a highly-structured Bach fugue, but gave unstable results for the other four scores, highlighting the greater flexibility of SIATEC-based algorithms in discovering inexactly repeating patterns. Finally, the exactly repeating structures identified by musicologist Siglind Bruhn within eight works from Bach's *Das Wohltemperierte Clavier* Book I were compared to rules within grammar-based models produced by ZZ for each piece. Although results showed wide variation, strong correlation existed at high levels of the hierarchy, a notable achievement considering that ZZ possesses no domain knowledge.

In conclusion, the results generally support the link between strength of compression and the information recovered, as suggested by the Minimum Description Length principle. The outcome shows that ZZ can outperform several popular compressors when applied to detect degradation in musical structure and classification of Dutch folk tunes, in the latter case when provided with attribute-rich note data. However, exact grammars cannot rival current techniques when seeking expert-defined patterns containing variations, and can fail to generate desirable rules where an overlapping explanation, and therefore rule, exists. The findings highlight the significance of intersection in the analysis of musical compositions, and support suggestions by existing studies that the ability to abstract musical features and pattern templates using domain knowledge is important to algorithmic analysis; such additions are likely to improve the performance of ZZ on these applications. Even without such enhancements, the results demonstrated that grammars are a tool that perform at a useful level in the field of music analysis.

The data that support the findings of Chapter 5 are openly available from Cardiff University via URL http://doi.org/10.17035/d.2020.0098047203.

10.1.1 Future Work

A number of possibilities exist for further development of this work, and the following directions are suggested:

- A heuristic could be designed which retains the sensitivity of grammar-based compressors to errors (degraded musical structure), but does not require exhaustive exploration of the search space. Such a heuristic might select only patterns which, when altered, allow significantly increased compression, or use high-level abstraction to structure the search and terminate branches unlikely to result in improvement on subsequent iterations. This may allow error detection by grammar to become a practical option.

- Investigation into reduction of the variation observed with increasing number of errors could be carried out, in an attempt to smooth and enhance error response, and to isolate false-negative conditions. Potentially, input pre-processing to higher level structures may produce a more consistent compressor response.

- From the observation that combining representations offers an accuracy gain when classifying the MTC, grammar-based compressors may be combined with other compression algorithms, such as those studied here, to produce a weighted output. Leveraging significant properties of individual compressors in this manner is likely to result in an overall improvement to the applications presented.

- Grammars may be augmented with parametric structures providing greater potential for modelling sequences that are repeated with variations, potentially using a modifier framework as presented in Chapter 7, making use of more advanced domain knowledge. The addition of simple flexible matching, perhaps in a form where encoding overhead does not prohibit its use, is likely to increase tune family classification success rate. Various matching schemes could be tested, from advanced alignment- or distance-based transforms to those incorporating specific domain knowledge, such as motif, rhythm or phrase templates. Existing feature-centred techniques, such as facets (Stober, 2011), viewpoints (Conklin, 2013a, 2013b; Conklin & Witten, 1995; Goienetxea et al., 2016), or form definitions such as described by Giraud and Staworko (2015), might be added to the construction process, and direct comparison made against these studies. It is reasonable to hypothesise that more compact grammars would result in improved performance for musical applications.

- A large-scale, methodical musicological study of the analysis and segmentation of a large corpus of digital scores would likely be a significant benefit to the research community. Ground truth, hierarchical definitions from various schools of analysis, including scale annotations, transformations and transitions might be used to develop more powerful and accurate algorithms for score compression and processing. This study observed additional notes prefixed to expert-identified segments in some cases; evaluation of how musically admissible such extensions may be is possible given expert consensus. Potentially, generalised grammars capable of covering a given form or style might be programmatically generated and used to optimise compression.

10.2 Grammars allowing Rule Modification

Chapters 6 and 7 investigated a method for constructing grammars which would allow instances of production rules to produce varying outputs when they were expanded, with the aim of allowing smaller encodings to be produced. In contrast to the approach of Siyari and Gallé (2017), the manner in which rule instances may change was not restricted to a single variation, but instead based upon transformations whose character would be encoded in the grammar.

The response of the scheme to a corpus of musical scores was first investigated, and compressed models built for 5521 inputs. To prevent excessive construction times, the number of transforms which may be considered was limited to a maximum of 50, the maximum difference between substrings which may be chosen as an approximate match set to 40%, and a checkpoint of 22 iterations per attempt to generate a grammar for a given rule was set, beyond which construction would only continue if any improvement in model size were already evident. Smaller models were generated for $< 25\%$ of inputs, which, although a minority, nonetheless represented 1455 pieces. Allowing the algorithm greater freedom by relaxing the constraints on number of transforms, or allowing construction to continue for longer before early termination, may potentially increase the count of smaller models, but this was not tested. Construction times were prohibitive, demonstrating the strong increase in complexity resulting from the inclusion of a second dimension of rule modifiers in the search space. Of the models which benefitted from the addition of rule modification, an average reduction in encoding size of 5% was observed, with a maximum reduction of 22% in the best case. The results showed that grammars which include rule modification are useful for at least a large minority of musical inputs in generating more compact models.

Next, grammars which may feature rule modifications were constructed for two pieces from the Johannes Kepler University Patterns Test Database (Johannes Kepler University, 2013), and the segments formed by their production rules used as input for the symbolic, polyphonic *Discovery of Repeated Themes & Sections* task from MIREX 2016. Despite consideration of all possible transformations within the bounds defined above, a reduced grammar encoding was only discovered for one piece from the database. However, a noticable improvement in performance was observed for the chosen metrics *establishment* and *occurrence*, due to the ability of the grammar's production rules to represent a greater number of more varied segments within the input. The addition of rule modification was sufficient to make the method the strongest performer over any of the tested algorithms. Again, it is possible that relaxation of the constraints may allow smaller grammars to be produced for other pieces from the database, which in turn may improve overall performance on the *Discovery of Repeated Themes & Sections* task, but this possibility is left for future work.

When grammars which include rule modification were applied to Classification of the Meertens Tune Collections by "tune family", using only chromatic pitch sequences from the collection's strophes, an improvement in classification performance was observed over grammars whose rules could not be modified. Of the 360 pieces in the collection, the addition of rule modification enabled smaller grammars to be constructed for only 4 inputs, however this was sufficient to produce a minor increase in classification accuracy from 0.858 to 0.875. Where inputs were combined in pairs, 3.69% of all generated grammars were smaller than their fixed-rule counterparts, and performance on the affected inputs was improved overall with 6 additional correct classifications representing $> 4.4\%$ of all inputs, supporting the hypothesis that a smaller model is a more accurate representation of a given score. So few pieces is, however, not a significant sample which may be considered representative of the population of scores, and so it is not possible to conclusively cite the experiment as a demonstration of the correctness of the hypothesis. Nonetheless, it is interesting to note that all models whose encodings were reduced by the addition of rule modification presented an increased accuracy during classification.

10.2.1 Future Work

The results of these experiments suggest further work would be beneficial in clarifying the role of transforms and more compact grammars in application to musicological tasks, and would provide an interesting indication of the possibilities the addition of rule modification presents to grammars. This study suggests the following candidates:

- The space of modifications which are explored during grammar construction could be steadily increased, to observe whether they enable smaller encodings to be produced for a greater proportion of inputs, and potentially for smaller models to be generated overall. A suite of different modification types may be developed, and customised to the content of different kinds of input data – DNA strings or specific written languages, for example – and the resulting search space may be explored and evaluated in an attempt to isolate a more effective method of traversal.

- The combination of modification types may be investigated, to discover whether such compound transforms are beneficial to the compactness of a grammar for a given type of input string. This study investigated only the use of individual transforms, seeking first to understand the search space involved without the additional worst-case complexity of 2^n the consideration of all transform combinations might introduce. It is not impossible that compound or even overlapping transforms may be useful in minimising $|G|$, and this possibility could be explored given the foundational work presented in this thesis.

- Instead of applying a modification to the output of a preceeding rule, thus transforming only its local indices, transforms may be designed which operate on global or offset positions, allowing the presence of a modifier, or even a rule, to affect symbols beyond its local boundaries. Such a scheme may allow an encoding similar to the point-cloud approach of COSIATEC (Meredith, David, 2013) to be produced, or a hybrid design which is able to sequentially replace non-terminals with associated substrings but also able to apply transformations globally at any given stage of expansion.

- A method might be developed which identifies an ideal selection of transforms for a given type or collection of input data, perhaps by exploration of the reduction they offer a grammar during construction, or by extraction of the characteristics of the data based on existing domain knowledge or statistical significance. This selection may itself be taken as a characteristic of the data, and may be applied exclusively during grammar construction to produce smaller encodings, and potentially strongly reduce grammar construction times.

- A theoretical and empirical exploration of the ramifications of storing the process required to correctly apply a transform in the expanding program may be conducted, with respect to practicality and Kolmogorov Complexity. The size of the program required to return the grammar to the desired form should be considered when evaluating the compactness of the model it represents, and this is significant to any scheme which must perform specific operations during decompression, such as that of Siyari and Gallé (2017), or even in selecting to output a terminal or non-terminal.

- The MIREX 2016 polyphonic *Discovery of Repeated Themes & Sections* and Classification of the Meertens Tune Collections by "tune family" tasks may be repeated, instead choosing construction parameters which allow more compact models to be generated for each piece in the respective collection. This might be achieved by selection of a specific group of transforms which apply strongly to the input data, or by more expensive exploration of a far larger space of available transforms. It is reasonable to hypothesise that notably more compact grammars would result in improved performance on these tasks.

- In order to further reduce the construction time of grammars for which rule modifications are considered during construction, parallel processing may be used to simultaneously explore a single level of the lattice of all possible transform combinations, in the worst case allowing all combinations in that level to be evaluated in the time required to consider all constituent combinations for a single set of transforms. Use of GPU-based processing using a language such as CUDA or OpenCL could enable a very large number of transforms to be tested simultaneously, potentially as many as the number of threads the particular hardware device allows, and this may lead to the discovery of smaller models within the search space.

In conclusion, the results demonstrate that more compact grammars are certainly possible when transforms which leverage natural patterns and processes in the data are considered as part of their encoding, but comparison between the scheme and existing grammar construction processes is challenging due to differences in the decompression process. Notably, consideration of different data transformations adds a new dimension to the construction process itself, introducing further complexity to its computation.

10.3 Improving the Speed of Lattice Traversal by Approximation of Constituent Gain

Chapter 9 investigated a method of reducing the number of nodes in the lattice which must be evaluated by MGP, during its traversal when constructing a grammar from a given input. The method employed bitmasks representing the graph nodes which were currently assigned to a production rule during consideration of each new candidate constituent, and those which remained free for use by the candidate's edges, spreading its possible gain across each relevant node to produce a linear approximation of the overall effect of including it as a production rule. Once approximations are known for all candidates, they may be tested for effectiveness by MGP, and the first offering any significant gain added to the encoding.

When applied without rule modifications to standard linguistic and DNA corpora, the method generated models of comparable size to ZZ, with compression ratios significantly better than those an IRR scheme could provide. The majority of models were equal in encoding length to those produced by ZZ, and for the purposes of algorithm comparison the bitmask method's models in this instance may be considered equivalent. Most significantly, the new method enabled a massive reduction in the number of MGPs necessary during construction, which in turn resulted in an exponential reduction in compressor run-time. The experiment confirmed the heuristic was effective when applied to linguistic and DNA data, and did not provide any evidence suggesting a limitation in relation to other input contexts. The approximation of candidate constituent gain using bitmasks was able to greatly improve construction times, even on large data, and the experiment provided evidence that constituent selection heuristics are useful in building grammars more efficiently.

When applied to 7448 pieces from the corpus of musical scores, again without rule modifications, the method continued to exhibit exponentially reduced grammar build times through a massive reduction in the number of MGPs required. However, on smaller inputs – in particular those for which no small encoding could be produced – construction times were greater than those of ZZ. Larger inputs of several hundred symbols or more gave the strongest results, showing the technique is most appropriate for sequences of this length. On average, grammar build times were reduced by 46.2%, with a $> 99\%$ reduction in the best case. However, the experiment noted a trade-off in achieving this performance, with $< 38\%$ of models being, on average, 0.8% larger than those generated by ZZ. The final node chosen by the algorithm from the lattice of candidate constituents was rarely equivalent; $> 68.8\%$ of grammars did not contain the same production rules, however $< 31.4\%$ of these were of equivalent encoding length. Overall, the majority of grammars produced using bitmask-based approximation of constituent gain were equal or smaller in size than their ZZ equivalents. Despite the strong decrease in computational complexity the technique allows, it cannot rival algorithms which operate in near-linear time, such as those of the IRR class, in terms of construction speed.

Rule modification was enabled, and grammars were again constructed using bitmask-based approximations of constituent gain for 5718 pieces from the corpus of musical scores. An average increase in construction speed of 22*x* was observed when generating models which included modifiers, and in 8% of cases smaller encodings were produced than those constructed using a rule modification-enabled version of ZZ. These models were 1.56 symbols smaller on average, and were generated 19.8*x* faster using the bitmask method. Nearly half of all models were less than or equal in encoding size to equivalent rule modification-enabled models constructed using ZZ, but over 65% were less than or equal in encoding size to those produced witout the benefit of rule modification. The remaining 35% were larger by an average of 6.92 symbols. Overall, model sizes were more similar to grammars which did not allow rule modification, showing the method in its current form is less suitable for producing smaller grammars which include rule modification, unless a likely increase in encoding size of approximately 3% is tolerable to the application. However, the technique is more often capable of producing smaller models than standard ZZ, showing it has potential for use as a quick construction method for encodings which include rule transforms.

10.3.1 Future Work

These results suggest a definite relationship between the quality of knowledge of each candidate constituent's effectiveness, the number of expensive graph parses required, and the computational complexity of the grammar construction algorithm. It is clear that development of a further improved method of candidate selection may be beneficial in the faster production of smaller, more inclusive models. The study suggests the following as useful or promising areas of investigation:

- The algorithm might be modified to prevent any significant degradation in construction speed for small inputs, or for those which cannot be compressed because no candidate production rules exist which would allow a more compact encoding than the length of the original input sequence to be produced. During the construction process, ZZ must first examine the effect of adding each candidate constituent to an empty set of production rules, in order to assess whether any may provide compression. For n candidates, n MGPs are needed. The initial step when constructing a grammar using bitmask-based approximation of constituent suitability is to perform n MGPs to calculate the maximum possible gain each constituent could provide. Where no positive gain is discovered, construction could be abandoned, giving an equivalent runtime to ZZ under the same circumstances. For very small inputs, a simple heuristic could be designed to prevent the construction process from continuing if insufficient substrings were first discovered. This would ensure algorithm run-times for the new method were always comparable or better than those of ZZ.

- Modes of failure which occur when constructing grammars using the bitmask-based method could be investigated, and a scheme designed which could prevent their occurrence. One specific mode which might be investigated is bitmask saturation – where so few bits remain unset that it becomes impossible for the simplistic scheme to provide accurate gain prediction – as this highlights that the technique is unable to assess gain where strong interactions between existing and candidate edges occur. In this instance, a metric could be developed to evaluate the degree of saturation at any stage of construction, and an alternative approximation allowed to progressively or categorically attain control where measurements show bitmasks are becoming ineffective. Design of a heuristic which specifically offers a reasonable approximation of edge interactions is likely to be easier to attain than a general scheme which retains accuracy across widely changing conditions, and may also provide greater insight into the ideal construction process through examination of alternative approximation methods.

- Perhaps in concert with the above suggestion, provision might be made within the construction algorithm to allow a fall back to ZZ if evaluation showed that a less optimal grammar were likely to be generated, as an option to allow for the output of grammars at least as compact as those generated by ZZ. Currently, the algorithm operates on a "bitmask only" basis, in contrast to a "ZZ only" approach. It is possible a parameter could be added which would allow tuning of construction, such that a desired proportion p of all iterations when adding a new constituent to the model would feature approximation, and $1-p$ would employ ZZ-style MGP steps to accurately assess the effect of the new constituent. This parameter might be automatically tuned by heuristic, such as in the above suggestion, so that compact grammars could be constructed more quickly than by ZZ alone. Adjustment of the search space explored by the algorithm could be beneficial in allowing a given application the choice between faster or smaller encodings.

In conclusion, the results demonstrate that it is possible to construct compact grammars with far lesser complexity than offered by schemes such as ZZ, providing an accurate estimation of constituent gain at each step can be made. However, the vastly-increased search space created by the addition of potential rule modifications means it is far more challenging to find an encoding which approaches that of a global optimum, and although the presented method of gain approximation dramatically decreases compression time there is a likelihood of creating models which are larger than desired when also considering rule modifications. However, an improved approximation scheme could potentially rectify the shortcoming.

10.4 Coda

This study has examined the possibility of applying grammars to a range of musicological tasks, and discovered that even a standard construction process is capable of reasonable success when seeking error corrections in musical scores, classifying scores by tune family, or segmenting them into musically-significant sections, despite having no built-in musical knowledge. This work also presented a powerful framework for allowing grammars whose rules are flexible to be generated, and showed it enabled the production of smaller models overall even with basic transforms, but that the search space of all possible flexibility represented another dimension which must be explored in addition to that of the candidate grammar rules. As a step towards practical generation of such models, a heuristic was developed which vastly reduced grammar construction times, and experiments showed approximation of constituent gain was a viable technique by which this may be achieved.

However, to keep construction times within practicable limits, it was necessary to constrain the number of rule modification types which were considered for each model, and to estimate whether a given construction iteration would fail to yield a reduction in encoding length so that such attempts may be aborted to improve overall compression times. It appears that a side-effect of such constriction is, naturally, the production of larger models. Continued investigation may provide a means by which build times and encoding lengths might be further optimised. However, at the conclusion of the current research it remains apparent that there is indeed an exponential relationship between the theoretically smallest grammar and the complexity involved in its discovery – eliminating the final few symbols in a grammar's encoding may indeed produce a more accurate representation of the input data, but it is perhaps attainable only at an impractical cost.

Nonetheless, the possibility of automatically exploring a universe of relevant transformations, resulting in a tremendous interweaving of patterns and permutations in the structured-but-fluid manner of a great composer, is a tantalising thought, and one which might aid the computation and understanding of structured creativity. It was the aim of this work to take a step closer to such a possibility, and alongside providing a useful, extensible contribution, it has been the author's privilege to have caught even a glimpse of this potential universe. Although the techniques and results presented here do not in themselves represent a definitive solution, they do offer considerable promise for the future, and add force to the assertion that musical sequences, in analysis or in experience, are most certainly a combination of many fascinating facets.

A

Grammar Encodings for Bach's Fugue No. 20 from WTC-II

Listing A.1: A human-readable display of the encoding of a grammar without rule modifications, constructed using ZZ from Bach's Fugue No. 20 from Das Wohltemperierte Clavier Book II. The encoding length is 583. The label TERM is used to represent unique termination symbols, and r<integer> is used to denote a production rule. Terminal symbols are chromatic pitch values. The grammar's start rule, S, is here represented as the first rule (r1). Rule lengths are specified at the end of each sequence.

```
r  1:  76 ,72 ,77 ,68 ,74 ,71 ,76 ,72 ,69 ,66 ,74 ,71 , r2 ,72 , r3 ,72 , r4 , r5 , r25 ,84 ,
       83 ,81 ,79 ,77 ,81 , r26 , r5 , r25 , r4 , r6 , r7 , r3 ,71 , r3 ,72 , r2 ,69 ,66 , r38 ,
       r37 ,79 ,84 ,75 ,81 ,78 ,83 ,79 ,76 ,73 ,81 , r8 , r5 , r26 ,72 , r5 ,79 ,81 ,82 ,
       81 , r26 , r6 , r6 , r27 ,62 ,60 , r6 ,64 ,69 ,57 ,62 ,77 ,71 ,74 ,68 ,71 , r25 ,69 ,
       73 ,76 ,79 , r8 , r6 , r4 ,67 ,77 ,76 ,81 ,77 ,82 ,73 ,79 ,76 ,81 , r8 , r8 ,79 ,76 ,
       72 ,69 , r8 ,71 ,68 ,76 , r6 , r37 ,83 ,81 ,80 ,78 , r25 , r25 , r27 , r34 , r17 ,65 ,
       64 , r38 ,64 ,68 ,69 ,128 , r7 ,72 ,63 ,69 ,66 ,71 , r2 ,61 ,69 ,65 , r9 , r2 ,69 ,
       68 ,57 , r34 , r28 , r10 , r35 ,57 ,59 , r30 ,54 ,56 ,56 ,54 ,56 ,57 , r18 , r29 ,55 ,
       55 , r18 , r31 ,65 ,65 , r19 , r34 ,62 , r19 , r28 ,55 ,60 ,65 , r32 ,62 ,65 ,62 ,67 ,
       60 ,57 , r9 ,59 ,61 , r20 ,62 , r6 ,78 ,71 , r25 ,69 , r6 , r7 ,66 ,67 , r7 ,66 ,64 ,
       63 ,64 ,66 ,64 ,63 , r20 ,63 ,63 ,61 ,63 ,64 , r9 , r2 , r19 , r34 ,65 ,59 ,62 , r39 ,
       r34 ,66 ,69 , r6 ,67 ,65 , r19 ,60 ,70 , r3 ,73 ,74 , r4 ,70 ,64 ,67 ,61 ,76 ,69 ,
          r32 ,74 ,67 , r9 ,68 , r3 , r34 ,60 , r9 ,57 ,56 ,57 ,64 , r35 ,54 , r35 , r11 ,51 ,
          52 , r10 ,62 , r20 ,62 ,65 ,56 , r10 ,64 ,129 ,64 , r39 ,57 ,54 ,63 ,52 ,64 , r29 ,
          r17 ,55 , r13 , r12 , r11 ,51 ,52 , r11 ,51 , r21 ,51 ,51 ,49 ,51 ,52 , r22 , r12 ,
          r23 ,50 , r22 , r23 ,48 , r14 , r23 ,52 ,53 ,53 ,47 ,50 ,44 ,47 ,52 ,45 , r13 , r22 ,
          r12 ,53 ,50 ,55 ,53 ,52 ,57 ,47 ,53 ,50 ,55 ,52 ,52 ,54 , r11 , r13 ,52 ,57 , r36 ,
          42 ,51 ,40 ,60 ,54 ,57 ,51 ,54 ,59 ,52 ,40 ,41 ,43 ,45 ,43 , r24 ,57 ,53 ,62 ,56 ,
          52 ,57 , r18 , r35 , r30 , r33 , r12 , r23 , r40 ,44 ,45 , r15 ,44 ,42 ,40 ,42 ,44 ,
          44 ,42 ,44 ,45 , r16 , r21 , r14 ,53 , r11 , r13 ,57 ,59 ,59 , r18 ,62 ,64 ,64 ,62 ,
          r19 ,69 , r7 ,69 ,70 ,58 ,60 ,62 ,60 , r31 ,55 ,57 , r31 , r22 , r23 ,49 , r14 ,50 ,
          49 , r15 ,47 ,49 , r21 , r14 , r40 ,43 ,45 ,47 , r15 , r16 ,60 ,62 ,64 , r28 ,59 ,
          r16 , r14 , r33 , r16 , r14 ,54 ,56 , r30 ,50 ,48 ,53 ,44 , r36 ,40 ,43 , r24 ,40 ,
          r24 ,40 ,33 ,45 ,TERM  (len=429)
r  2:     67 ,64 ,TERM  (len=3)
r  3:     69 ,71 ,TERM  (len=3)
r  4:     72 ,74 ,TERM  (len=3)
r  5:     76 ,77 ,TERM  (len=3)
r  6:     72 ,71 ,TERM  (len=3)
r  7:     69 ,67 ,TERM  (len=3)
r  8:     77 ,74 ,TERM  (len=3)
r  9:     62 ,59 ,TERM  (len=3)
r10 :     59 ,60 ,TERM  (len=3)
r11 :     54 ,52 ,TERM  (len=3)
r12 :     57 ,55 ,TERM  (len=3)
r13 :     54 ,55 ,TERM  (len=3)
r14 :     50 ,52 ,TERM  (len=3)
r15 :     47 ,45 ,TERM  (len=3)
r16 :     47 ,48 ,TERM  (len=3)
r17 :     60 ,59 ,57 ,TERM  (len=4)
r18 :     57 , r10 ,TERM  (len=3)
r19 :     64 ,65 ,67 ,TERM  (len=4)
r20 :     61 ,59 ,61 ,TERM  (len=4)
r21 :     49 ,47 ,49 ,TERM  (len=4)
r22 :     52 ,53 ,55 ,TERM  (len=4)
r23 :     53 ,52 ,50 ,TERM  (len=4)
```

```
r24 :  41 ,40 ,38 ,TERM  ( len =4)
r25 :  76 ,74 , r6 ,TERM  ( len =4)
r26 :  79 ,77 ,76 ,74 ,TERM  ( len =5)
r27 :  69 ,68 ,66 ,64 ,TERM  ( len =5)
r28 :  65 ,64 ,62 ,60 ,TERM  ( len =5)
r29 :  62 , r17 ,TERM  ( len =3)
r30 :  57 ,56 , r11 ,TERM  ( len =4)
r31 :  58 , r12 ,53 ,TERM  ( len =4)
r32 :  69 ,62 ,65 ,59 ,TERM  ( len =5)
r33 :  50 ,48 , r15 ,TERM  ( len =4)
r34 :  r7 ,65 ,64 ,62 ,TERM  ( len =5)
r35 :  r29 ,56 ,TERM  ( len =3)
r36 :  50 ,47 ,52 ,48 ,45 ,TERM  ( len =6)
r37 :  r4 ,76 ,78 ,80 ,81 ,TERM  ( len =6)
r38 :  66 ,68 ,68 ,66 ,68 , r3 ,TERM  ( len =7)
r39 :  60 ,65 ,56 , r9 ,64 ,60 ,TERM  ( len =7)
r40 :  r23 ,48 , r16 , r33 ,TERM  ( len =5)

Grammar  size :  583
```

Listing A.2: A human-readable display of the encoding of a grammar including rule modifications, constructed using ZZ from Bach's Fugue No. 20 from Das Wohltemperierte Clavier Book II. The encoding length is 411; 172 symbols less than the encoding shown above. The label TERM is again used to represent unique termination symbols, r<integer> is used to denote a production rule, and m<integer> is used to denote a modifier. Where a modifier should be applied to a given rule, its index follows the prime symbol as a suffix to the rule, and expansion of the rule should occur before the transform is applied. For instance, "r13'2" states that the expansion of rule 13 should be transformed by modifier 2. Within a modifier's encoding, "TRANS,x" means "translate the rule by x", "DEL,x" means "delete the symbol at position x", and "REV" means "reverse the entire sequence". Terminal symbols are chromatic pitch values. The grammar's start rule, S, is here represented as the first rule (r1). Rule and modifier lengths are specified at the end of each sequence. Within the encoding, a single special symbol separates the rule and modifier sequences.

```
r  1:  76 , r29 , r13 ‘2 ,71 ,72 , r28 ‘8 ,84 ,83 ,81 ,79 ,77 , r14 ‘8 ,76 , r30 , r19 ,
       r34 ‘6 , r20 , r29 ‘6 ,77 ,74 , r17 ‘4 ,72 ,76 , r28 ‘4 ,74 , r2 , r2 ,69 , r16 ‘8 , r2 ,
       64 ,69 ,57 ,62 , r12 , r14 ‘6 ,73 ,76 ,79 ,77 , r24 ‘6 ,81 , r25 ‘4 ,77 ,74 ,79 ,
       r13 ,71 ,68 ,76 ,72 , r20 ,83 , r20 ‘9 , r14 ‘6 ,68 ,66 ,64 , r14 , r5 , r3 , r18 ‘6 ,
       71 ,64 ,68 ,69 ,128 ,69 , r29 ‘2 ,65 , r4 ,67 ,64 ,69 , r31 , r32 , r33 , r27 ‘6 ,55 ,
       r5 ‘9 ,58 , r7 ,65 ,65 , r26 ‘8 , r27 ‘8 ,55 ,60 ,65 , r9 ,62 ,65 ,62 ,67 , r34 , r2 ,
       78 ,71 , r14 ‘6 , r2 ,69 , r32 ‘6 , r33 ‘6 , r4 ,67 ,64 , r26 ‘8 ,65 ,59 ,62 , r25 ‘7 ,
       r14 ,66 ,69 , r2 , r24 ,71 ,73 ,74 ,72 ,74 ,70 ,64 ,67 ,61 ,76 ,69 , r9 ,74 ,67 ,
       r4 ,68 ,69 ,71 , r14 ,60 , r4 ,57 ,56 ,57 , r14 ‘2 ,56 ,54 ,62 , r5 , r6 ,51 ,52 ,59 ,
       60 ,62 , r10 ,65 ,56 ,59 ,60 ,64 ,129 ,64 , r25 ,54 , r31 ‘2 , r32 ‘2 , r33 ‘2 , r26 ,
       r27 , r8 ,52 , r8 , r21 ,54 ,55 ,52 ,53 ,55 , r7 ,50 ,55 ,53 ,52 ,57 ,47 ,53 ,
       r34 ‘3 ,52 ,57 , r15 ,42 ,51 ,40 , r21 ‘6 ,40 , r22 ,57 ,53 ,62 ,56 ,52 ,57 ,57 ,
       r17 ,56 ,57 , r16 ,47 ,45 , r7 ,52 ,50 , r30 ‘1 ,44 ,45 ,47 ,45 , r36 ‘2 ,48 ,49 ,
       47 ,49 , r8 , r35 ,62 ,64 , r35 ‘5 , r22 ‘4 , r23 , r32 ‘3 ,50 , r36 , r30 ‘1 , r19 ‘1 ,
       r28 , r26 ‘2 , r20 ‘1 , r16 ,53 ,44 , r15 ,40 ,43 , r11 , r11 ,33 ,45 ,TERM
       ( len =284)
r  2:  72 ,71 ,TERM  ( len =3)
r  3:  65 ,64 ,TERM  ( len =3)
r  4:  62 ,59 ,TERM  ( len =3)
r  5:  60 ,59 ,57 ,TERM  ( len =4)
r  6:  56 ,54 ,52 ,TERM  ( len =4)
r  7:  57 ,55 ,53 ,TERM  ( len =4)
r  8:  50 ,52 ,53 ,TERM  ( len =4)
r  9:  69 ,62 ,65 ,59 ,TERM  ( len =5)
r10 :  61 ,59 ,61 ,62 ,TERM  ( len =5)
r11 :  41 ,40 ,38 ,40 ,TERM  ( len =5)
r12 :  77 ,71 ,74 ,68 ,71 ,TERM  ( len =6)
r13 :  76 ,72 ,69 ,77 ,74 ,TERM  ( len =6)
r14 :  69 ,67 , r3 ,62 ,TERM  ( len =5)
r15 :  50 ,47 ,52 ,48 ,45 ,TERM  ( len =6)
r16 :  r6 ,50 ,48 ,TERM  ( len =4)
r17 :  59 ,60 ,62 , r5 ,TERM  ( len =5)
r18 :  59 ,61 , r10 ,TERM  ( len =4)
r19 :  67 , r18 ‘5 ,TERM  ( len =4)
r20 :  71 ,72 ,74 ,76 ,78 ,80 ,81 ,TERM  ( len =8)
r21 :  r12 ‘1 ,52 ,45 ,TERM  ( len =5)
r22 :  41 ,43 , r14 ‘1 ,TERM  ( len =5)
r23 :  r7 ‘9 ,58 , r7 ,TERM  ( len =5)
r24 :  67 , r3 ,65 ,67 ,60 ,70 ,69 ,TERM  ( len =8)
r25 :  60 ,65 ,56 , r15 ‘8 ,TERM  ( len =6)
```

```
r26:  52,r22‘8,TERM (len=4)
r27:  r8,55,r8‘9,48,TERM (len=6)
r28:  r23‘6,59,TERM (len=4)
r29:  r25‘8,66,74,TERM (len=5)
r30:  77,76,74,72,r17‘8,TERM (len=7)
r31:  68,57,r14,r3,62,TERM (len=6)
r32:  60,r17,56,57,59,TERM (len=6)
r33:  57,r6,r18‘2,TERM (len=5)
r34:  60,57,r4,r18,TERM (len=5)
r35:  54,52,54,55,57,59,59,r5‘9,TERM (len=10)
r36:  r33‘3,52,TERM (len=4)

Modifiers in use:
m 1:  TRANS,−24,TERM (len=3)
m 2:  TRANS,−5,TERM (len=3)
m 3:  TRANS,−7,DEL,1,TERM (len=5)
m 4:  TRANS,17,TERM (len=3)
m 5:  TRANS,10,TERM (len=3)
m 6:  TRANS,7,TERM (len=3)
m 7:  DEL,8,TERM (len=3)
m 8:  TRANS,12,TERM (len=3)
m 9:  REV,TERM (len=2)

Grammar size: 411
```

B

A worked example of constructing a Transform-enabled Grammar

The following description is a real-world example of the construction process for a transform-enabled grammar, showing how repeating terms and their associated transforms may be selected, and how a choice may be made between constituents for inclusion in the encoding. For the sake of clarity, only specific constituents and transforms are considered in this example, but the resulting grammar is consistent with that produced when all possible elements are considered. Where a transition is being discussed, such as the addition of a constituent to an existing set, the notation $C_{current}$ and C_{next} are used to differentiate between states. Where a single state only is relevant, it is simply notated as C, but may refer to the current state or a changed state, as indicated by the description. A node within the lattice of all possible constituents is represented as $C = \{c_1, c_2, \ldots, c_n\}$, and a node within the lattice of all possible transforms is represented as $M = \{T_1, T_2, \ldots, T_n\}$. Where encodings are specified, all termination symbols \$ should be considered unique, e.g. $\$_1, \$_2, \ldots, \$_n$.

To begin, the grammar is initialised by setting S to the input sequence, in this case the chromatic pitch values of "*NLB073516_01*" from the Meertens Tune Collections (Meertens Instituut, 2018).The tune may be seen in Figure B.1, as it appears in the *MTC-ANN v2.0.1* archive. This piece is 41 symbols long, and an additional termination symbol, \$, is appended to S:

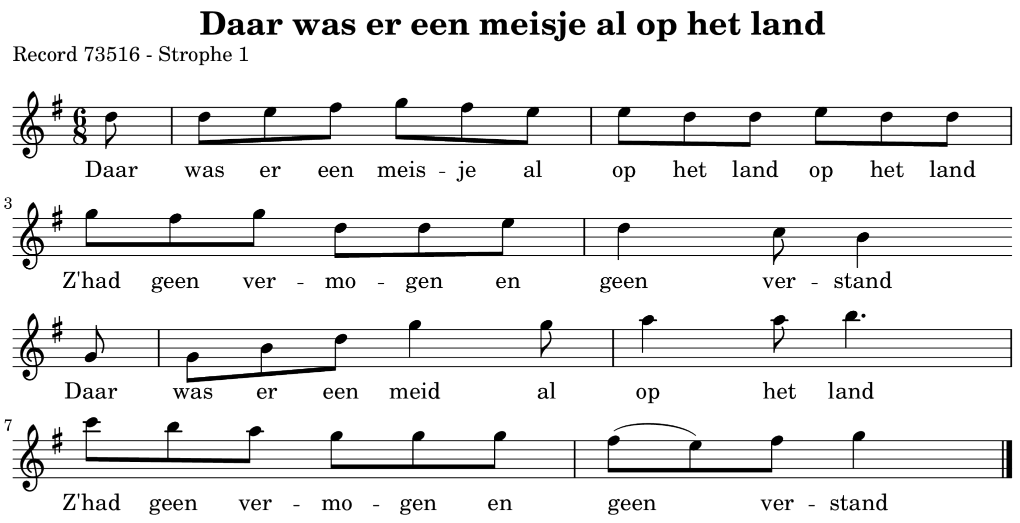

Daar was er een meisje al op het land
Record 73516 - Strophe 1
Daar was er een meis - je al op het land op het land
Z'had geen ver - mo - gen en geen ver - stand
Daar was er een meid al op het land
Z'had geen ver - mo - gen en geen ver - stand

Figure B.1: Music notation for the tune *Daar was er een meisje al op het land*, piece "*NLB073516_01*" from the Meertens Tune Collections (Meertens Instituut, 2018), as it appears in the *MTC-ANN v2.0.1* archive.

G := 74, 74, 76, 78, 79, 78, 76, 76, 74, 74, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, 79, 79, 81, 81, 83, 84, 83, 81, 79, 79, 79, 78, 76, 78, 79, \$

A search is performed over all substrings within S, using Algorithm 8, with the set of all possible transforms including *Reverse* and *Translate*, the latter with a range which covers $-127 : 127$. It is most sensible to choose the set of translations to consider based on all pairwise intervals in S, or to compute them individually for each combination a, b during the search, but for the purposes of this example "all transforms" may be taken as $\{[reverse], [translate, -127], [translate, -126] \dots [translate, -1], [translate, 1] \dots [translate, 126], [translate, 127], none\}$. Algorithm 8 combines *Reverse* with all other transforms; in practice, a combination of all possible transform types and parameters may be more appropriate where the properties of any patterns embedded in the data are unknown, to prevent the exclusion of potentially useful transforms.

Performing the search with $minLength = 2, maxLength = 10, maxDistance = 3$ results in selection of 114 candidate terms of lengths $2 \leq l \leq 10$, with 405 occurrences throughout S. Table B.1 shows a tiny subset of these matches.

Once all candidate constituents for the current encoding of G have been chosen, combinations of these may be selected and Minimal Grammar Parsing used to obtain the smallest possible encoding given each combination, and the model minimising $|G|$ retained; the constituent search may then be

Algorithm 8 Find all exact and flexible repeats, with associated transforms. Assumes T[1] = reverse

```
Require: $S$, $minLength$, $maxLength$, $allTransforms$, $maxDistance$
Ensure: $D$, (dictionary of all exact and approximate repeats)
 1: for $l \leftarrow minLength$ to $maxLength$ do
 2:     for $i \leftarrow 1$ to $length(S) - l$ do
 3:         $a \leftarrow S[i : i + l]$
 4:         for $m \leftarrow minLength$ to $maxLength$ do
 5:             for $j \leftarrow 1$ to $length(S) - m$ do
 6:                 $b \leftarrow S[j : j + m]$
 7:                 for $r \leftarrow 0$ to $1$ do        ▷ Test $a$ normally ($r \leftarrow 0$) and reversed ($r \leftarrow 1$)
 8:                     if $r = 1$ then
 9:                         $c \leftarrow f(T[1], a$                  ▷ Apply the reverse transform to $a$
10:                     else
11:                         $c \leftarrow a$                  ▷ Do not apply the reverse transform to $a$
12:                     end if
13:                     for $k \leftarrow 2$ to $length(allTransforms)$ do
                                                              ▷ Step through all other transforms
14:                         $c, d \leftarrow f(T[k], c$
15:                         if $d \leq maxDistance$ AND $a = c$ then     ▷ Record all matches
                                                              occurring within the chosen distance
16:                             if doesn't exist $D[a]$ then
17:                                 $D[a][1] = [i, l]$
18:                             end if
19:                             $D[a][\text{ end }] = [j, m, T[k]$
20:                         end if
21:                     end for
22:                 end for
23:             end for
24:         end for
25:     end for
26: end for
```

Constituent	Term a	Index i	Term b	Index j	Transform T	Distance d
1	$[74, 74]$	1	$[74, 74]$	1	none	0
			$[74, 74]$	9	none	0
			$[74, 74]$	12	none	0
			$[74, 74]$	17	none	0
2	$[74, 76]$	2	$[74, 76]$	2	none	0
			$[74, 76]$	10	none	0
			$[74, 76]$	18	none	0
3	$[74, 74, 76, 78, 79, 78, 76, 76, 74, 74]$	1	$[74, 74, 76, 78, 79, 78, 76, 76, 74, 74]$	1	none	0
			$[79, 79, 81, 81, 83, 84, 83, 81, 79, 79]$	27	$T = [reverse, translate, 5]$	3
4	$[74, 76, 74, 74, 79, 78, 79, 74, 74, 76]$	10	$[74, 76, 74, 74, 79, 78, 79, 74, 74, 76]$	10	none	0
			$[76, 74, 74, 79, 78, 79, 74, 74, 76, 74]$	11	$T = [reverse]$	1
5	$[79, 79, 81, 81, 83, 84, 83, 81, 79, 79]$	27	$[79, 79, 81, 81, 83, 84, 83, 81, 79, 79]$	27	none	0
			$[74, 74, 76, 78, 79, 78, 76, 76, 74, 74]$	1	$T = [reverse, translate, -5]$	3

Table B.1: A small subset of string pairs a, b found in *NLB073516_01* during a flexible substring search

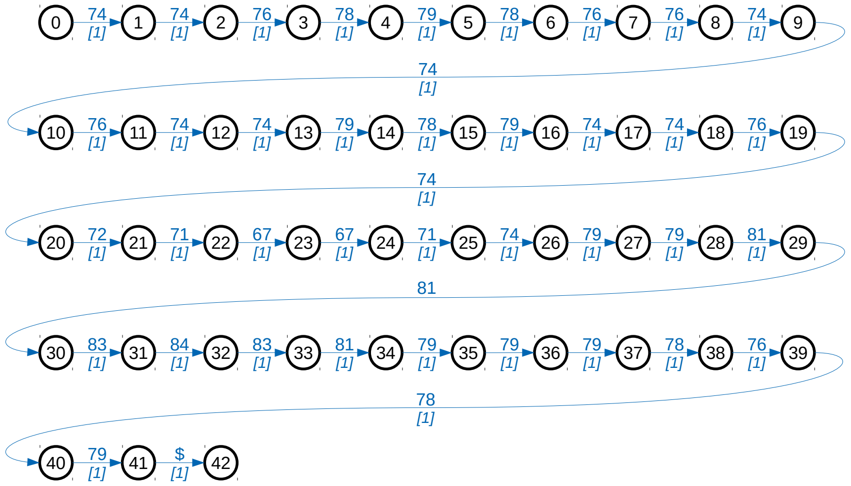


Figure B.2: Parse graph for *S*, with $C = \emptyset, \mathcal{M} = \emptyset$. Taking a blue edge between any two circular nodes adds the upper symbol to the encoding of *G*, and costs the value shown beneath the edge in brackets. Units are in symbols. Since $C = \emptyset$, there are no constituent sub-graphs to parse, only that of the input, *S*.

repeated for the new encoding of *G* if required. During parsing, graphs for both the existing *G* and any constituents being newly considered are parsed, so that those which form sub-constituents may be leveraged to generate a smaller, hierarchical encoding. ZZ activates or deactivates exactly one element from the current combination at each iteration, thus exploring the lattice of all possible combinations as described in Section 4.7, and the node producing the greatest reduction in $|G|$ is moved to when the iteration ends, in preparation for the next.

The process can begin with any *C* and $\mathcal{M}$, although Carrascosa et al. (2010; 2011; 2012) show that choosing $> 1/3n$ constituents, where *n* is the number of all possible candidate constituents, cannot provide a minimal $|G|$. The choice of initial state, and exact order of candidate selection, determines the local minima which will be reached. For this example, the initial state will be set to $C = \emptyset, \mathcal{M} = \emptyset$.

For each change $C_{current} \neq C_{next}$ or $\mathcal{M}_{current} \neq \mathcal{M}_{next}$, graphs are generated for the current encoding of *G* and constituents in *C*, including the candidate which is to be evaluated. The graph for $C = \emptyset, \mathcal{M} = \emptyset$ may be seen in Figure B.2.

Since at most only one arriving and one departing edge exists for each node in the graph, there are no alternative paths. Traversal from start to end nodes requires 42 steps, thus $|G| = 42$, and results in encoding $G = S$ as shown above, and initial $|G_{min}| = 42$. ZZ begins in the *top-down* phase, exploring *C* nodes connected with the current one, within the level immediately beneath it.

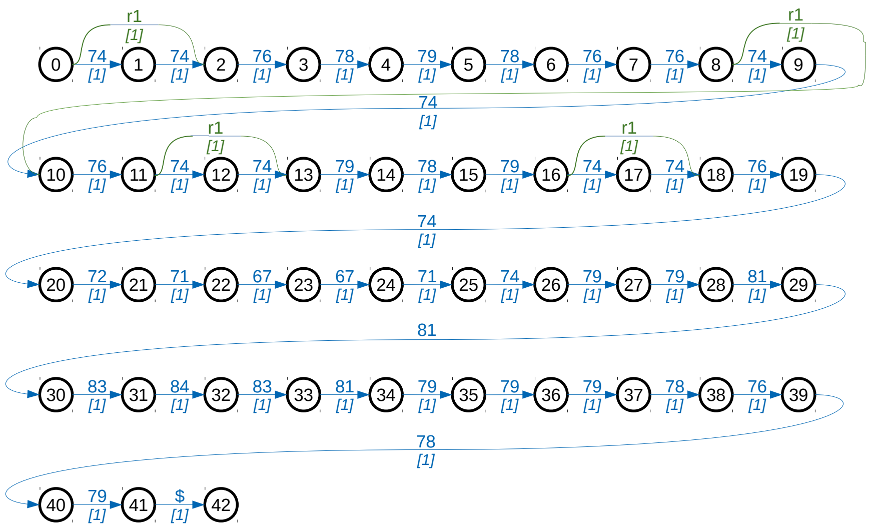


Figure B.3: Parse graph for S, with $C = \{c_1\}, \mathcal{M} = \emptyset$. Additional edges, marked in green, are created from the inclusion of constituent 1.

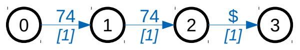


Figure B.4: Parse graph for constituent 1, with $C = \{c_1\}, \mathcal{M} = \emptyset$.

For simplicity, constituents not listed in Table B.1 are omitted from this example, but the results are unchanged. Each constituent is added in turn to the empty set $C_{current}$ to produce C_{next}, graphs are generated given the temporarily changed C, and a minimal parse is made over each graph and constituent sub-graph. The total number of nodes visited during parsing is equal to the minimal encoding size $|G|$ for that temporary combination of C. The graphs constructed for $C = \{c_1\}, \mathcal{M} = \emptyset$ are shown in Figures B.3 – B.4. All b terms for constituent 1 may be used even with $\mathcal{M} = \emptyset$, as no transforms are required.

The shortest route through the graph for S now uses each edge added by the inclusion of constituent 1. A minimal parse from start to end nodes for both the graph for S and that for constituent 1 results in $|G| = 38 + 3 = 41$, and the following concatenated encoding:

$G := r1, 76, 78, 79, 78, 76, 76, r1, 76, r1, 79, 78, 79, r1, 76, 74, 72, 71, 67, 67, 71, 74, 79, 79, 81, 81, 83, 84, 83, 81, 79, 79, 79, 78, 76, 78, 79, \$, 74, 74, \$$

This process is repeated for $C = \{c_2\}$, resulting in $|G| = 42$ since its edges allow for a gain of 3

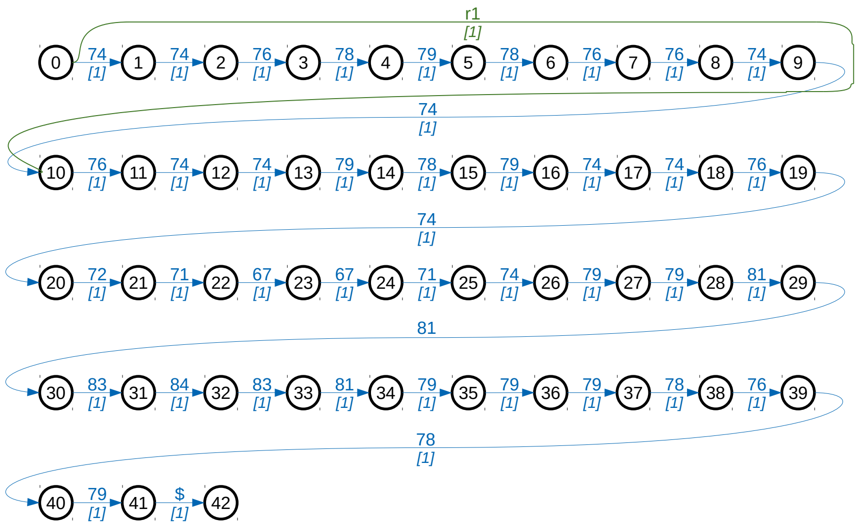


Figure B.5: Parse graph for *S*, with $C = \{c_3\}, \mathcal{M} = \emptyset$. Only one edge, marked in green, is created from the inclusion of constituent 3.

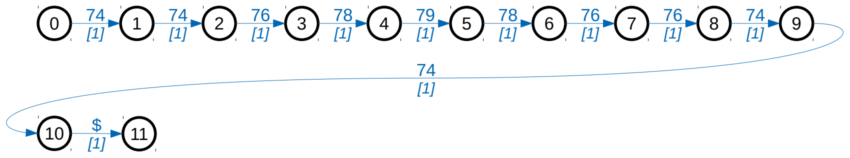


Figure B.6: Parse graph for constituent 3, with $C = \{c_3\}, \mathcal{M} = \emptyset$.

symbols, but the constituent requires 3 symbols to be added to the encoding G.

When $C = \{c_3\}, \mathcal{M} = \emptyset$ is evaluated, the second edge cannot be used since it relies on a transform $T \notin \mathcal{M}$. Wherever an edge has an unsatisfied transform dependency, it is omitted from the parse graph, thus "de-activating" it during the MGP. In this instance, the graphs in Figures B.5 – B.6 are generated:

Taking the shortest path through these graphs results in the following encoding:

$G \;:=\; r1, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, 79, 79, 81, 81, 83, 84, 83, 81, 79, 79, 79, 78, 76, 78, 79, \$, 74, 74, 76, 78, 79, 78, 76, 76, 74, 74, \$$

The same is true for $C = \{c_4\}$ and $C = \{c_5\}$, which both require a transform $T \notin \mathcal{M}$ to provide any gain. Table B.2 shows the results of considering the addition of exactly one constituent to $C = \emptyset$, thus exploring every connected node in the C lattice on the level immediately beneath node $C = \emptyset$.

C	$\|G\|$
G_{min}	42
$\{c_1\}$	41
$\{c_2\}$	42
$\{c_3\}$	44
$\{c_4\}$	44
$\{c_5\}$	44

Table B.2: Encoding sizes resulting from adding a single constituent to $C = \emptyset$ with $\mathcal{M} = \emptyset$

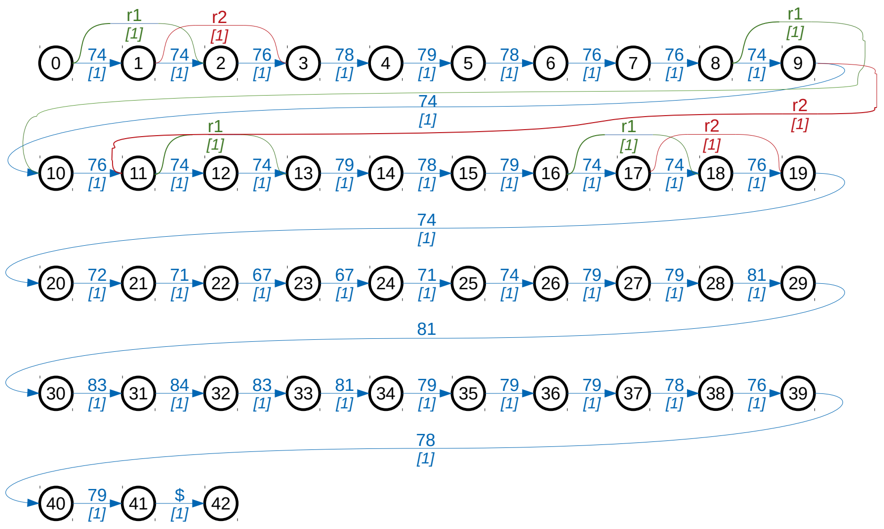


Figure B.7: Parse graph for S, with $C = \{c_1, c_3\}, \mathcal{M} = \emptyset$. Edges for constituent 1 are marked in green, and edges for constituent 3 in red.

The result of this ZZ step is selection of $C = \{c_1\}$, since $|G_{C=\{c_1\},\mathcal{M}=\emptyset}| < |G_{min}|$ where $G_{C=\{c_1\},\mathcal{M}=\emptyset}$ indicates the grammar resulting from the addition of constituent 1 to C, and no change to $\mathcal{M}$. Thus, the new smallest model becomes $G_{C=\{c_1\},\mathcal{M}=\emptyset}$ with $|G_{min}| = 41$. Traversal through the constituent lattice may now continue from the node $C = \{c_1\}$.

Once more, the addition of each candidate constituent to C which is not already present in it must be evaluated. The first new evaluation is for constituent 2, with $C = \{c_1, c_2\}$, producing the graphs shown in Figures B.7 – B.9.

The shortest path through the graph for S makes use of all constituent 1 edges and excludes all constituent 2 edges. There are no options for a shorter parse of the sub-graphs for the individual

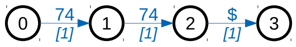


Figure B.8: Parse graph for constituent 1, with $C = \{c_1, c_3\}, M = \emptyset$.

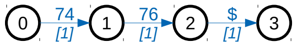


Figure B.9: Parse graph for constituent 3, with $C = \{c_1, c_3\}, M = \emptyset$.

constituents, and so the following encoding results, with $|G| = 44$, identical to that for $C = \{c_1\}$ but with constituent 2 appended:

$G :=$ *r*1, 76, 78, 79, 78, 76, 76, *r*1, 76, *r*1, 79, 78, 79, *r*1, 76, 74, 72, 71, 67, 67, 71, 74, 79, 79, 81, 81, 83, 84, 83, 81, 79, 79, 79, 78, 76, 78, 79, \$, 74, 74, \$, 74, 76, \$

Once again, since $M = \emptyset$, constituents 3 – 5 cannot offer any gain, and are simply appended to the encoding at additional cost. Table B.3 shows the results of exploring all nodes connected with, and belonging to the level immediately beneath, the node $C = \{c_1\}$.

Satisfying $|G_{C=\{c_1,c\},M=\emptyset}| < |G_{min}|$ has been impossible for any new candidate c considered during this step, and so removal of a constituent from C is tested instead, to discover whether a reduction in $|G|$ occurs as part of ZZ's *bottom-up* phase. The only possibility is $C = \emptyset$ giving $|G| = 42$, which also cannot satisfy $|G_{C=\{c_1\}\setminus\{c\},M=\emptyset}| < |G_{min}|$. ZZ has reached a node in the C lattice from which $|G|$ cannot improve, and the final encoding for $M = \emptyset$ is generated from $C = \{c_1\}$. Where no transforms are available, the mimimum encoding possible is $|G_{min}| = 41$.

All nodes in the M lattice immediately below $M = \emptyset$ can now be explored. In this example, the set of all usable transforms consists of $T_1 = [reverse, translate, 5]$, $T_2 = [reverse]$, $T_3 = [reverse, translate, -5]$. ZZ moves to the node $M = \{T_1\}$, and traversal of the C lattice begins again,

C	$\|G\|$
G_{min}	41
$\{c_1, c_2\}$	44
$\{c_1, c_3\}$	52
$\{c_1, c_4\}$	52
$\{c_1, c_5\}$	52

Table B.3: Encoding sizes resulting from adding a single constituent to $C = \{c_1\}$ with $M = \emptyset$

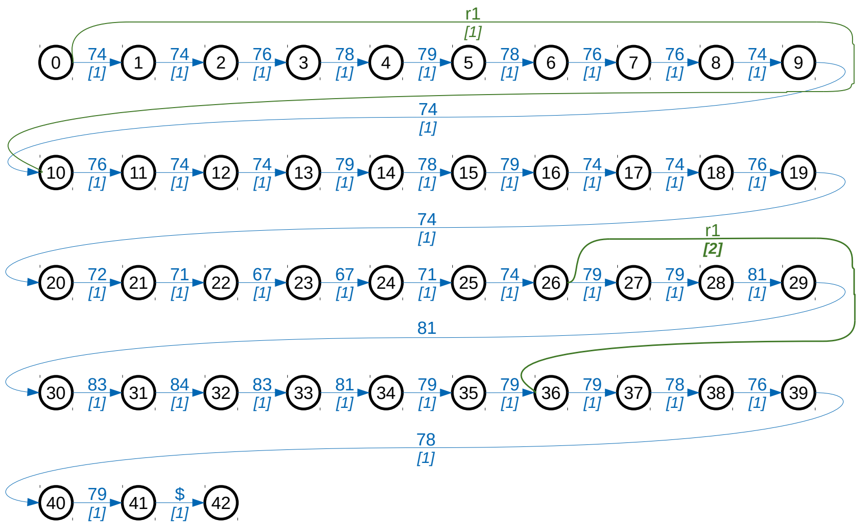


Figure B.10: Parse graph for S, with $C = \{c_3\}, \mathcal{M} = \{T_1\}$. Note the additional edge between nodes 26 and 36, with cost 2, made possible by the inclusion of T_1

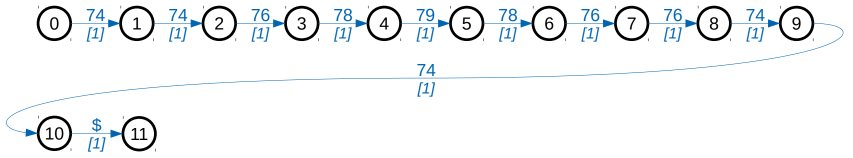


Figure B.11: Parse graph for constituent 3, with $C = \{c_3\}, \mathcal{M} = \{T_1\}$.

but this time without evaluating $C = \emptyset$, because inclusion of any $T_n \in \mathcal{M}$ in the original encoding cannot by definition result in $|G_{C=\emptyset,\mathcal{M}\neq\emptyset}| < |G_{C=\emptyset,\mathcal{M}=\emptyset}|$.

The first evaluations are made for $C = \{c_1\}$ and $C = \{c_2\}$. Grammar sizes for both are already known, since constituents from the set $\{c_1, c_2\}$ are not associated with any T. The next evaluation is for $C = \{c_3\}$, where constituent 3 does have $T \in \mathcal{M}$, and the edge using T can be included in the parse graphs, shown in Figures B.10 – B.11.

A minimal parse of these graphs results in $|G| = 40$, with T_1 appended in the encoding of G, and symbol ! added to separate the production rules from the transforms:

G := *r*1, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, *r*1, *t*1, 79, 78, 76, 78, 79, \$, 74, 74, 76, 78, 79, 78, 76, 76, 74, 74, !, *reverse*, *translate*, 5, \$

C	$\lvert G\rvert$
G_{min}	41
$\{c_1\}$	45
$\{c_2\}$	48
$\{c_3\}$	40
$\{c_4\}$	56
$\{c_5\}$	56

Table B.4: Encoding sizes resulting from adding a single constituent to $C = \emptyset$ with $\mathcal{M} = \{T_1\}$

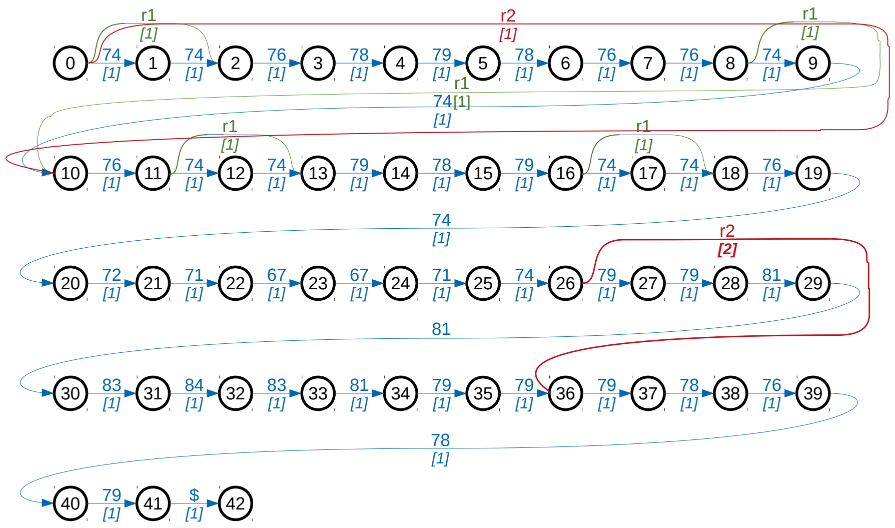


Figure B.12: Parse graph for S, with $C = \{c_1, c_3\}, \mathcal{M} = \{T_1\}$. Edges for constituent 1 are marked in green, and edges for constituent 3 in red.

Grammar size for $C = \{c_4\}$ and $C = \{c_5\}$ is also known, since both require $T \notin \mathcal{M}$ to activate any new graph edges and offer a potential reduction of $|G|$. Table B.4 shows the results of exploring all nodes in the constituent lattice immediately beneath $C = \emptyset$ for $\mathcal{M} = \{T_1\}$.

The result of this ZZ step is selection of $C = \{c_3\}$, since $|G_{C=\{c_3\},\mathcal{M}=\{T_1\}}| < |G_{min}|$. Thus the grammar $G_{C=\{c_3\},\mathcal{M}=\{T_1\}}$ is obtained, with $|G_{min}| = 40$. Traversal through the constituent lattice may now continue from the node $C = \{c_3\}$. The first addition to C is constituent 1, giving $C = \{c_1, c_3\}$, producing the graphs shown in Figures B.12 – B.14.

Minimally parsing these graphs requires the use of all edges in S belonging to constituent 3 and some belonging to constituent 1, and both edges in the graph for constituent 3 belonging to con-

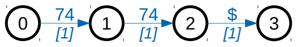


Figure B.13: Parse graph for constituent 1, with $C = \{c_1, c_3\}, \mathcal{M} = \{T_1\}$.

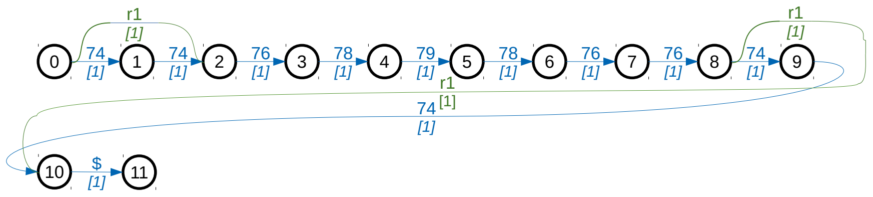


Figure B.14: Parse graph for constituent 3, with $C = \{c_1, c_3\}, \mathcal{M} = \{T_1\}$. Note the additional constituent 4 edges which may now be used during parsing, which exist due to the inclusion of constituent 1 in G.

stituent 1. The second edge belonging to constituent 3 requires T_1, and so its weight is 2, representing the inclusion of two non-terminals, one referencing the rule generated by constituent 3, the other referencing the transform T_1 which must be applied to it. This parse produces the following encoding, with $|G| = 39$:

G := $r2, 76, r1, 79, 78, 79, r1, 76, 74, 72, 71, 67, 67, 71, 74, r1, t1, 79, 78, 76, 78, 79, \$, 74, 74, \$,$ $r1, 76, 78, 79, 78, 76, 76, r1, !,$ *reverse*, *translate*, $5, \$$

The next evaluation is for $C = \{c_2, c_3\}$. An edge from constituent 2 can be used to minimise the parsing of constituent 3, but only once, and constituent 2 has fewer edge instances within the graph for S. As such, it cannot improve the current $|G_{min}|$. Constituents 4 and 5 require a $T \notin \mathcal{M}$ to offer any gain, and so also cannot improve $|G_{min}|$. Table B.5 shows the results for this ZZ step.

The result of this ZZ step is selection of $C = \{c_1, c_3\}$, with the new $|G_{min}| = 39$. Traversal through the C lattice may again continue from this node. Since $T \notin \mathcal{M}$ would be required to allow

C	$\|G\|$
G_{min}	40
$\{c_1, c_3\}$	39
$\{c_2, c_3\}$	41
$\{c_3, c_4\}$	51
$\{c_3, c_5\}$	51

Table B.5: Encoding sizes resulting from adding a single constituent to $C = \{c_3\}$ with $\mathcal{M} = \{T_1\}$

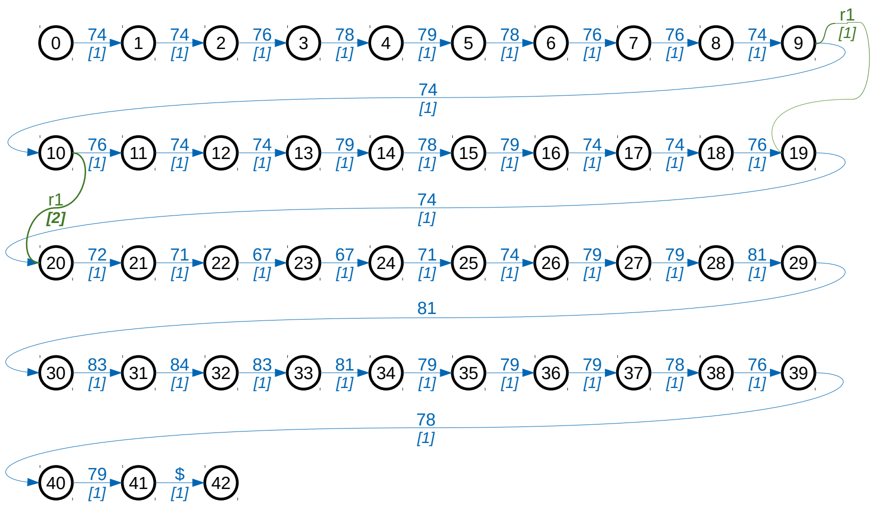


Figure B.15: Parse graph for S, with $C = \{c_4\}, \mathcal{M} = \{T_2\}$, including a second edge belonging to constituent 4 which is enabled by the presence of $T_2 \in \mathcal{M}$.

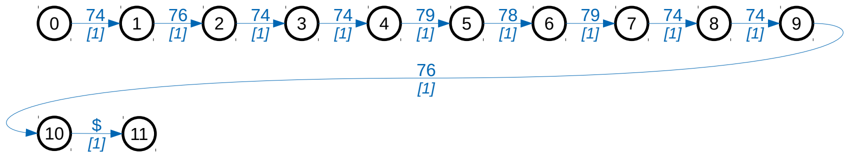


Figure B.16: Parse graph for constituent 4, with $C = \{c_4\}, \mathcal{M} = \{T_2\}$.

constituents 4 and 5 to reduce $|G|$, and all edges from constituent 2 overlap with more useful edged from constituent 1 and do not contribute to the shortest path, no further reduction in $|G|$ can occur. This is also true when constituents are removed from C in the upwards ZZ step. As such, the new model is $G_{C=\{c_1,c_3\},\mathcal{M}=\{T_1\}}$, giving $|G_{min}| = 39$.

Next, the $\mathcal{M}$ lattice node for $\mathcal{M} = \{T_2\}$ is evaluated. Since only constituent 4 makes use of T_2, the resulting encoding size for all other constituents with $\mathcal{M} = \{T_2\}$ is known, as $|G_{C=\{c\},\mathcal{M}=\emptyset]}| + |T_2| + 1$ where $c \in \{c_1, c_2, c_3, c_5\}$. The parse graphs for $C = \{c_4\}$ are shown in Figures B.15 – B.16.

The shortest path through these graphs makes use only of the edge from constituent 4 which does not require a transform, as both its edges overlap and there is an additional symbol of overhead for choosing the second edge. The following encoding results, with T_2 remaining unused:

$$G \quad := \quad 74, 74, 76, 78, 79, 78, 76, 76, 74, r1, 74, 72, 71, 67, 67, 71, 74, 79, 79, 81, 81, 83, 84, 83, 81,$$

C	$\lvert G\rvert$
G_{min}	39
$\{c_1\}$	43
$\{c_2\}$	44
$\{c_3\}$	46
$\{c_4\}$	46
$\{c_5\}$	46

Table B.6: Encoding sizes resulting from adding a single constituent to $C = \emptyset$ with $\mathcal{M} = \{T_2\}$

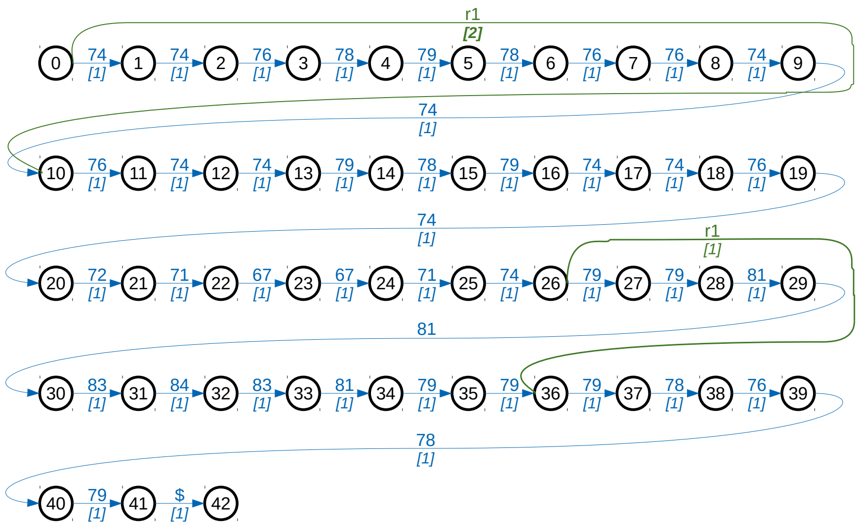


Figure B.17: Parse graph for S, with $C = \{c_5\}, \mathcal{M} = \{T_3\}$. Note the strong similarity with Figure B.11; identical edges exist, but here it is the first edge which carries the additional cost required by the use of a transform.

79, 79, 79, 78, 76, 78, 79, \$, 74, 76, 74, 74, 79, 78, 79, 74, 74, 76, !, *reverse*, \$

Table B.6 shows the results for this final ZZ step for $\mathcal{M} = \{T_2\}$, and G_{min} remains unchanged:

Since T_3 is leveraged only by constituent 5, which has an inverse relationship to constituent 3 with respect to transformation, the same results are found during the next ZZ traversal of the $\mathcal{M}$ lattice with $\mathcal{M} = \{T_3\}$, when a single constituent is added to $C = \emptyset$. The parse graphs for $C = \{c_5\}$ are shown in Figures B.17 – B.18.

The resulting encoding is very similar to that seen for $C = \{c_3\}, \mathcal{M} = \{T_1\}$, but with the negative translation applied to the first instance of the offset rule:

$$G \quad := \quad r1, t1, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, r1, 79, 78, 76, 78, 79, \$,$$

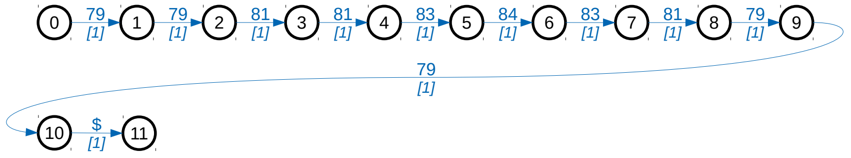


Figure B.18: Parse graph for constituent 5, with $C = \{c_5\}, \mathcal{M} = \{T_3\}$.

C	$\lvert G\rvert$
G_{min}	39
$\{c_1\}$	45
$\{c_2\}$	46
$\{c_3\}$	48
$\{c_4\}$	48
$\{c_5\}$	40

Table B.7: Encoding sizes resulting from adding a single constituent to $C = \emptyset$ with $\mathcal{M} = \{T_3\}$

79, 79, 81, 81, 83, 84, 83, 81, 79, 79, !, *reverse*, *translate*, −5, $

Table B.7 shows the results for this ZZ step.

However, only two instances of the sequence [79, 79] exist in the input S, and only two instances of the term [74, 74] from constituent 1 are now present in the encoding, reducing its ability to minimise $|G|$. As such, the subsequent ZZ step adding constituent 1 to $C = \{c_5\}, \mathcal{M} = \{T_3\}$ cannot produce the same minimisation as $C = \{c_3\}, \mathcal{M} = \{T_1\}$. Instead, an encoding of length 41 symbols is generated, and $|G|$ cannot be reduced. ZZ traversal for $\mathcal{M} = \{T_3\}$ ends, and G_{min} remains unchanged. Table B.8 shows the results for this final ZZ step for $\mathcal{M} = \{T_3\}$, and G_{min} remains unchanged:

Having evaluated all $\mathcal{M}$ lattice nodes immediately beneath the node for $\mathcal{M} = \emptyset$, $\mathcal{M} = \{T_1\}$ is chosen as offering the smallest value for $|G|$ during a ZZ C lattice traversal. The traversal of the $\mathcal{M}$

C	$\lvert G\rvert$
G_{min}	39
$\{c_1, c_5\}$	41
$\{c_2, c_5\}$	42
$\{c_3, c_5\}$	51
$\{c_4, c_5\}$	51

Table B.8: Encoding sizes resulting from adding a single constituent to $C = \{c_5\}$ with $\mathcal{M} = \{T_3\}$

C	$\lvert G\rvert$
G_{min}	39
$\{c_1\}$	47
$\{c_2\}$	48
$\{c_3\}$	42
$\{c_4\}$	50
$\{c_5\}$	50

Table B.9: Encoding sizes resulting from adding a single constituent to $C = \emptyset$ with $M = \{T_1, T_2\}$

C	$\lvert G\rvert$
G_{min}	39
$\{c_1, c_3\}$	41
$\{c_2, c_3\}$	43
$\{c_3, c_4\}$	44
$\{c_3, c_5\}$	53

Table B.10: Encoding sizes resulting from adding a single constituent to $C = \{c_3\}$ with $M = \{T_1, T_2\}$

lattice may now continue, with a single unused transform added to $M = \{T_1\}$ as each node immediately below is evaluated, beginning with $C = \emptyset, M = \{T_1, T_2\}$. Table B.9 shows the known results for the first ZZ step with $M = \{T_1, T_2\}$, which is $|G_{C=\{c\},M=\{T_1\}}| + |T_2| + 1$ where $c \in \{c_1, c_2, c_3, c_5\}$. Since both edges from constituent 4 cannot be used together, it is already known that T_2 cannot offer any reduction in $|G|$ for $C = \{c_4\}$ and its use would simply add to the encoding length.

The C lattice node presenting a smallest $|G|$ is chosen in conclusion to this ZZ step, resulting in $C = \{c_3\}$, and traversal continues with the evaluation of all nodes immediately below it. Since no transforms apply to constituents 1 and 2, and $T_3 \notin M$, encoding sizes here are simply $|G_{C=\{c_3,c\},M=\{T_1\}}| + |T_2| + 1$ for $c \in \{c_1, c_2, c_4, c_5\}$. Transforms applying to constituents 3 and 4, however, are available. Although the first edge of constituent 4 overlaps with a more optimal edge from constituent 3 and therefore is unused, its second edge, which requires T_2, can be used, since this edge is included in the graph given $T_2 \in M$. The associated parse graphs are shown in Figures B.19 – B.21.

A minimal parse results in $|G| = 44$, as shown in the encoding:

$G := r1, r2, 72, 71, 67, 67, 71, 74, r1, t1, 79, 78, 76, 78, 79, \$, 74, 74, 76, 78, 79, 78, 76, 76, 74, 74, \$, 74, 76, 74, 74, 79, 78, 79, 74, 74, 76, !, \mathit{reverse}, \mathit{translate}, 5, \$, \mathit{reverse}, \$$

The results for this ZZ step are shown in Table B.10.

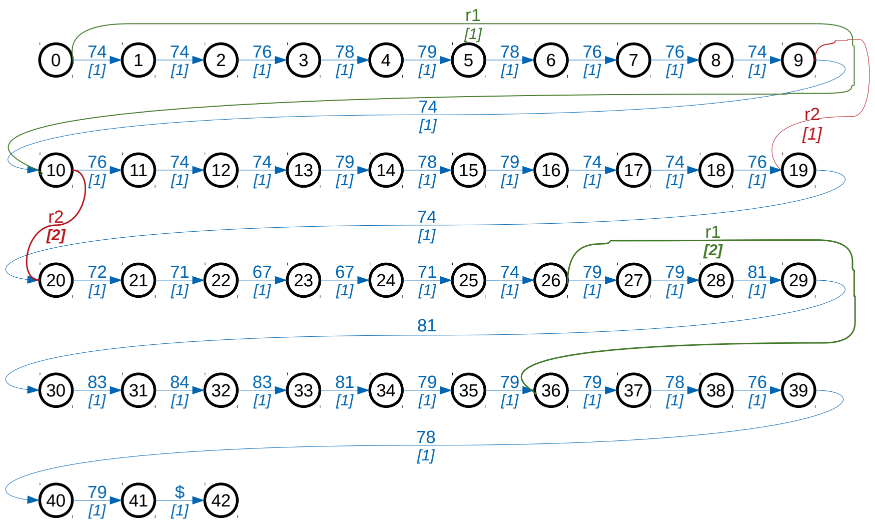


Figure B.19: Parse graph for S, with $C = \{c_3, c_4\}, \mathcal{M} = \{T_1, T_2\}$. Edges for constituent 1 are marked in green, and edges for constituent 3 in red.

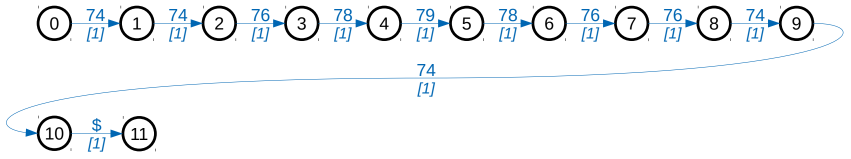


Figure B.20: Parse graph for constituent 3, with $C = \{c_3, c_4\}, \mathcal{M} = \{T_1, T_2\}$.

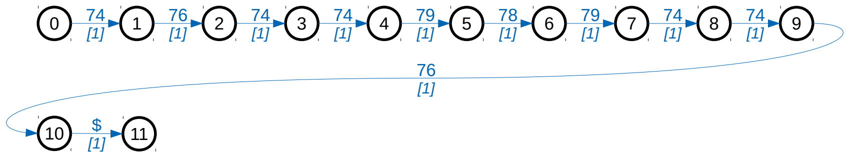


Figure B.21: Parse graph for constituent 4, with $C = \{c_3, c_4\}, \mathcal{M} = \{T_1, T_2\}$.

C	$\lvert G\rvert$
G_{min}	39
$\{c_1\}$	45
$\{c_2\}$	46
$\{c_3\}$	44
$\{c_4\}$	48
$\{c_5\}$	44

Table B.11: Encoding sizes resulting from adding a single constituent to $C = \emptyset$ with $\mathcal{M} = \{T_1, T_3\}$

For the sake of brevity, it can be stated that no further improvements to $|G|$ are possible for $\mathcal{M} = \{T_1, T_2\}$, either by adding or removing constituents against C. As such, the minimum G for this $\mathcal{M}$ is $C = \{c_1, c_3\}$ with $|G| = 41$, and so the minimal grammar $G_{C=\{c_1,c_3\},M=\{T_1\}}$ remains unchanged.

The next step in traversal of the $\mathcal{M}$ lattice is to add T_3 to $\mathcal{M}$. Doing so simply adds additional encoding overhead to all known $|G|$ where T_3 offers no benefit, resulting generally in $|G_{C=\{c\},M=\{T_1,T_3\}}| > |G_{min}|$ where $c \in c_1, c_2, c_3, c_4$. Inclusion of T_3 allows constituent 5 to offer some reduction in encoding length; parse graphs for $C = \{c_5\}$ with $\mathcal{M} = \{T_1, T_3\}$ are the same as for $\mathcal{M} = \{T_3\}$, as shown in Figures B.17 – B.18, due to the fact that transform encodings are not parsed. The shortest path through these graphs makes use of all constituent 5 edges, and produces the following encoding with $|G| = 44$:

G := *r*1, *t*1, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, *r*1, 79, 78, 76, 78, 79, \$, 79, 79, 81, 81, 83, 84, 83, 81, 79, 79, !, *reverse*, *translate*, 5, \$, *reverse*, *translate*, −5, \$

Table B.11 shows the results for this ZZ step.

Since $|G_{C=\{c_3\},M=\{T_1,T_3\}}|$ and $|G_{C=\{c_5\},M=\{T_1,T_3\}]}| < |G_{C=\emptyset,M=\emptyset}|$, the C lattice node for $C = \{c_3\}$ or $C = \{c_5\}$ may be chosen as producing the smallest $|G|$; for this example, the first is selected. Traversal continues to the nodes immediately beneath it, $C = \{c_3, c\}$ where $c \in \{c_1, c_2, c_4, c_5\}$. Minimal graph parsing results in the following encodings for $c \in \{c_1, c_2, c_4\}$:

Constituent 1: G := *r*2, 76, *r*3, 79, 78, 79, *r*1, 76, 74, 72, 71, 67, 67, 71, 74, *r*2, *t*1, 79, 78, 76, 78, 79, \$, 74, 74, \$, *r*1, 76, 78, 79, 78, 76, 76, *r*1, !, *reverse*, *translate*, 5, \$, *reverse*, *translate*, −5, \$

Constituent 2: G := *r*2, 76, 74, 74, 79, 78, 79, 74, *r*1, 74, 72, 71, 67, 67, 71, 74, *r*2, *t*1, 79, 78, 76, 78, 79, \$, 74, 76, \$, 74, *r*1, 78, 79, 78, 76, 76, 74, 74, !, *reverse*, *translate*, 5, \$, *reverse*, *translate*, −5, \$

Constituent 4: G := *r*1, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, *r*1, *t*1, 79, 78, 76, 78, 79, \$, 74, 74, 76, 78, 79, 78, 76, 76, 74, 74, \$, 74, 76, 74, 74, 79, 78, 79, 74, 74, 76, !, *reverse*, *translate*, 5, \$, *reverse*, *translate*, −5, \$

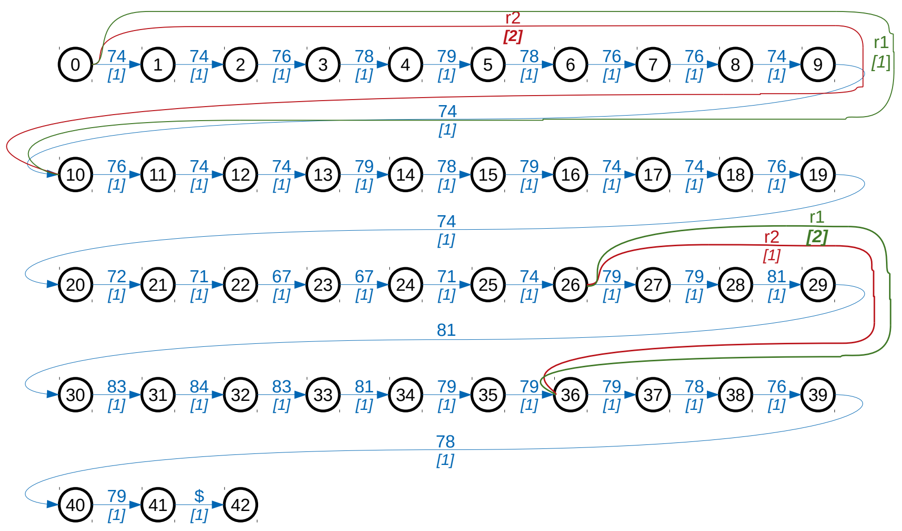


Figure B.22: Parse graph for S, with $C = \{c_3, c_5\}, \mathcal{M} = \{T_1, T_3\}$. Edges for constituent 3 are marked in green, and edges for constituent 5 in red.

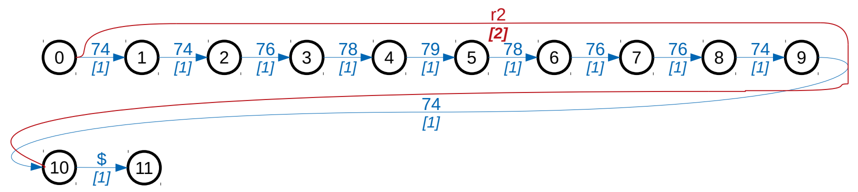


Figure B.23: Parse graph for constituent 3, with $C = \{c_3, c_5\}, \mathcal{M} = \{T_1, T_3\}$. Note that it is possible to take the edge offered by constituent 5 using T_3 to traverse the entire graph at a total cost of 3 symbols.

However, an interesting phenomenon occurs when adding constituent 5 to $C = \{c_3\}$. Since both constituents share an inverse relationship, it is possible to parse the graph for either constituent using the other's term combined with a transform, resulting in neither containing an encoding by which the other may be resolved. This problem of *term cancellation* may be solved relatively easily: during parsing, a record is kept of the constituents c_{child} whose fully-spanning edge has been used during the parsing of seen constituents c_{parent}, and all c_{parent} edges are disabled when c_{child} constituents are parsed, thus preserving at least one copy of each constituent term which may be converted entirely into another using an available transform.

To illustrate this, consider the parse graphs for $C = \{c_3, c_5\}$, shown in Figures B.22 – B.24.

The shortest path through the graph for constituent 3 takes the edge from node 0 to node 10, at

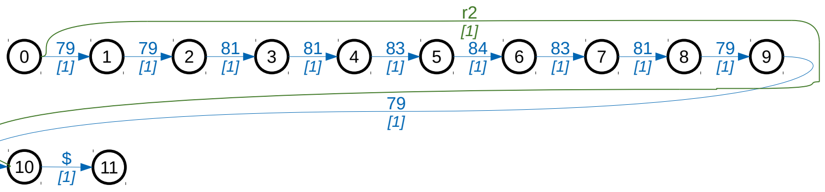


Figure B.24: Parse graph for constituent 5, with $C = \{c_3, c_5\}, \mathcal{M} = \{T_1, T_3\}$. Note that it is again possible to traverse the entire graph using the edge from constituent 3 and T_1; however, this is *invalid* if constituent 5 has already been used to traverse constituent 3.

C	$\|G\|$
G_{min}	39
$\{c_1, c_3\}$	43
$\{c_2, c_3\}$	45
$\{c_3, c_4\}$	55
$\{c_3, c_5\}$	47

Table B.12: Encoding sizes resulting from adding a single constituent to $C = \{c_3\}$ with $\mathcal{M} = \{T_1, T_3\}$.

a cost of 2 symbols; the alternative would be to take all nodes n_i to n_{i+1} until node 10, at a cost of 10 symbols. When constituent 5 is parsed, the same situation occurs, and the following invalid encoding is produced with $|G| = 43$:

G := *r1*, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, *r1*, *t1*, 79, 78, 76, 78, 79, \$, *r2*, *t2*, \$, *r1*, *t1*, !, *reverse*, *translate*, 5, \$, *reverse*, *translate*, -5, \$

Instead, using the technique described above to prevent *term cancellation*, a record is made that constituent 3 makes use of a fully spanning edge from constituent 5, and when the parse of constituent 5's graph is made the edge belonging to constituent 3 is disabled, either by its removal or by setting an edge cost which cannot be optimal (so that it will not be chosen when the shortest path is sought). The graphs parsed during this process will be as shown in Figures B.25 – B.27.

This time, the following valid encoding will result, with $|G| = 47$:

G := *r1*, 76, 74, 74, 79, 78, 79, 74, 74, 76, 74, 72, 71, 67, 67, 71, 74, *r1*, *t1*, 79, 78, 76, 78, 79, \$, *r2*, *t2*, \$, 79, 79, 81, 81, 83, 84, 83, 81, 79, 79, !, *reverse*, *translate*, 5, \$, *reverse*, *translate*, -5, \$

It can be seen that selecting two constituents with such a relationship is not optimal, and the smallest encoding is instead produced for $\mathcal{M} = \{T_1, T_3\}$ by $C = \{c_1, c_3\}$. Table B.12 shows the results for this ZZ step.

Since the existing minimum $|G|$ cannot be improved on this step, or by removal of any single

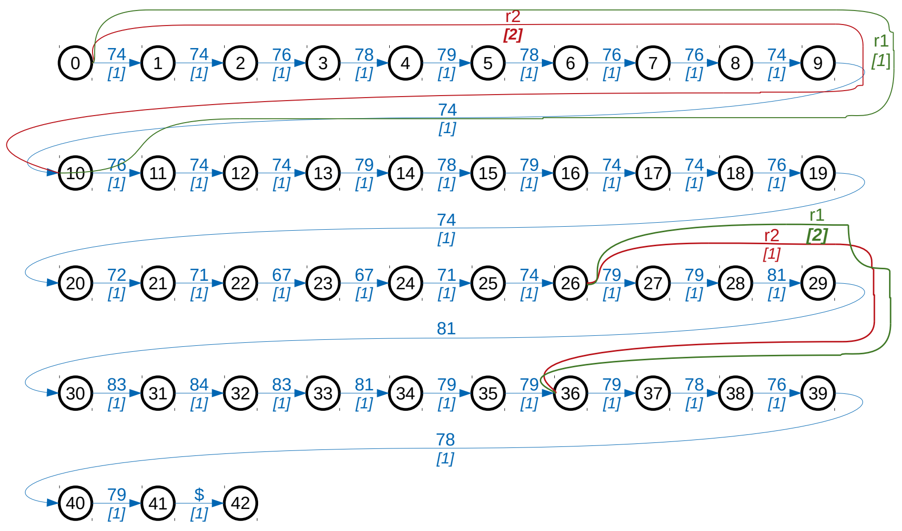


Figure B.25: Parse graph for *S*, with $C = \{c_3, c_5\}, \mathcal{M} = \{T_1, T_3\}$. Prevention of *term cancellation* does not change the edges available in this graph.

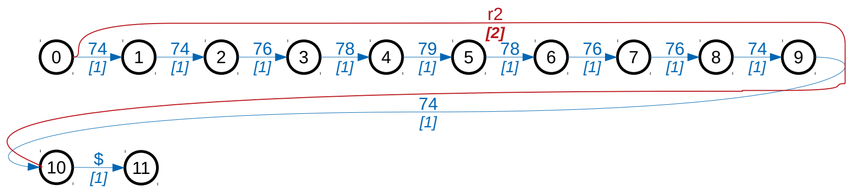


Figure B.26: Parse graph for constituent 3, with $C = \{c_3, c_5\}, \mathcal{M} = \{T_1, T_3\}$. Note that it is still possible to take the edge offered by constituent 5 using T_3 to traverse the entire graph.

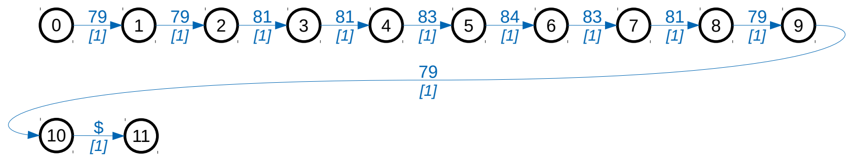


Figure B.27: Parse graph for constituent 5, with $C = \{c_3, c_5\}, \mathcal{M} = \{T_1, T_3\}$. Since a constituent 5 edge has been used during the parsing of constituent 3, it is *disabled* for this graph to prevent *term cancellation*.

constituent from C, the ZZ traversal of C completes. Traversal of the $\mathcal{M}$ lattice also completes, since addition of a second transform to $\mathcal{M} = \{T_1\}$ does not yield a smaller value for $|G|$. Thus, local minimisation of $|G|$ given $C, \mathcal{M}$ has been achieved, and the final ZZ-optimised grammar $C = \{c_1, c_3\}, \mathcal{M} = \{T_1\}$ with $|G| = 39$ is the compact model returned.

Providing *term cancellation* is avoided, real-world strings of moderate length may be parsed in a practical length of time. It is important to note that a combination of the possibilities C and $\mathcal{M}$ represents a very large search space, and as such further optimisation is necessary to enable fast generation of a smallest grammar or modelling of an input string S where $|S| \gg 1$.

References

Abdallah, S., Gold, N., & Marsden, A. (2016). Analysing Symbolic Music with Probabilistic Grammars. In *Computational Music Analysis* (pp. 157–189). Springer.

Apel, W. (1961). *The notation of polyphonic music, 900-1600* (No. 38). Medieval Academy of Amer.

Apel, W. (2003). *The Harvard Dictionary of Music*. Harvard University Press.

Apostolico, A., & Lonardi, S. (1998). Some theory and practice of greedy off-line textual substitution. In *Proceedings dcc'98 data compression conference (cat. no. 98tb100225)* (pp. 119–128).

Arnold, D., Arnold, D. M., & Scholes, P. A. e. a. (1983). *The new oxford companion to music* (Vol. 1). Oxford; New York: Oxford University Press.

Arnold, R., & Bell, T. (1997). A corpus for the evaluation of lossless compression algorithms. In *Proceedings dcc'97. data compression conference* (pp. 201–210).

Avid Technology Inc. (2011). *Sibelius 7: Using the ManuScript language.* Retrieved from http://resources.avid.com/SupportFiles/Sibelius/ManuScript_Language_Guide_2018.6.pdf (Accessed: 21-09-2018)

Bainbridge, D. (1996). An extensible Optical Music Recognition System. In *Proceeding of the 19^< th> australasian computer science conference, 1996.*

Bainbridge, D., & Bell, T. (2001). The Challenge of Optical Music Recognition. *Computers and the Humanities*, *35*(2), 95–121.

Baxter, J. H. (1931). *An Old St. Andrews Music Book: Cod. Helmst. 628: Published in Facsimile with an Introduction*. Humphrey Milford.

Bellert, I. (1965). Relational phrase structure grammar and its tentative applications. *Information and Control*, *8*(5), 503–530.

Benz, F., & Kötzing, T. (2013). An Effective Heuristic for the Smallest Grammar Problem. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation* (pp. 487–494).

Bernstein, L. (1976). *The Unanswered Question: Six talks at Harvard* (Vol. 33). Harvard

University Press.

Berry, D. A. (1996). *Statistics: a Bayesian perspective* (Vol. 4). Duxbury Press.

Bhate, S., & Kak, S. (1991). Pāṇini's grammar and computer science. *Annals of the Bhandarkar Oriental Research Institute*, *72*(1/4), 79–94.

Boot, P., Volk, A., & de Haas, W. B. (2016). Evaluating the Role of Repeated Patterns in Folk Song Classification and Compression. *Journal of New Music Research*, *45*(3), 223–238.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. e. a. (2000). *Extensible markup language (xml) 1.0*. W3C recommendation October.

Bruhn, S. (1993). *JS Bach's Well-Tempered Clavier: In-depth Analysis and Interpretation*. Mainer, Hong Kong.

Burlet, G., Porter, A., Hankinson, A., & Fujinaga, I. (2012). Neon.js: Neume Editor Online. In *Ismir* (pp. 121–126).

Burrows, M., & Wheeler, D. J. (1994). A block-sorting Lossless Data Compression Algorithm. *SRC Research Report 124*.

Byrd, D., & Simonsen, J. G. (2015). Towards a standard testbed for optical music recognition: Definitions, metrics, and page images. *Journal of New Music Research*, *44*(3), 169–195.

Callon, G. J. (1998-2009). *Acadia Early Music Archive.* Retrieved from `http://www.acadiau.ca/~gcallon/www/archive/` (Accessed: 20-09-2018)

Cambouropoulos, E. (2001). The Local Boundary Detection Model (LBDM) and its application in the study of Expressive Timing. In *ICMC* (p. 8).

Cambouropoulos, E., Crawford, T., & Iliopoulos, C. S. (2001). Pattern Processing in Melodic Sequences: Challenges, Caveats and Prospects. *Computers and the Humanities*, *35*(1), 9–21.

Caplin, W. E. (2001). *Classical form: A theory of formal functions for the instrumental music of haydn, mozart, and beethoven*. Oxford University Press.

Carpenter, P. (1967). The musical object. *Current Musicology*(5), 56–87.

Carrascosa, R., Coste, F., Gallé, M., & Infante-Lopez, G. (2010). Choosing Word Occurrences for the Smallest Grammar Problem. In *International Conference on Language and Automata Theory and Applications* (pp. 154–165).

Carrascosa, R., Coste, F., Gallé, M., & Infante-Lopez, G. (2011). The Smallest Grammar Problem as Constituents Choice and Minimal Grammar Parsing. *Algorithms*, *4*(4), 262–284.

Carrascosa, R., Coste, F., Gallé, M., & Infante-Lopez, G. (2012). Searching for Smallest Grammars on Large Sequences and Application to DNA. *Journal of Discrete Algorithms*, *11*, 62–72.

Chaitin, G. J. (1966). On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, *13*(4), 547–569.

Chambers, J. (2015). *John Chambers' clone of the O'Neill's Project files and web pages.* Retrieved from `http://trillian.mit.edu/~jc/music/book/oneills/1850` (Accessed: 25-09-2018)

Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., & Shelat, A. (2005). The Smallest Grammar Problem. *IEEE Transactions on Information Theory*, *51*(7), 2554–2576.

Chomsky, N. (1956). Three Models for the Description of Language. *IRE Transactions on information Theory*, *2*(3), 113–124.

Chomsky, N. (1959). On certain Formal Properties of Grammars. *Information and Control*, *2*(2), 137–167.

Cilibrasi, R., Vitányi, P., & de Wolf, R. (2004). Algorithmic Clustering of Music based on String Compression. *Computer Music Journal*, *28*(4), 49–67.

Clark, S., & Rehding, A. e. a. (2001). *Music theory and natural order from the renaissance to the early twentieth century.* Cambridge University Press.

Collins, T. (2016). *MIREX 2016: Discovery of Repeated Themes and Sections Results.* Retrieved from `https://www.music-ir.org/mirex/wiki/2016:Discovery_of_Repeated_Themes_&_Sections_Results` (Accessed: 30-09-2019)

Collins, T., & Meredith, D. (2013). Maximal Translational Equivalence Classes of Musical Patterns in Point-Set Representations. In *International Conference on Mathematics and Computation in Music* (pp. 88–99).

Conklin, D. (2013a). Fusion Functions for Multiple Viewpoints. In *Proceedings of the 6th International Workshop on Machine Learning and Music, Prague, Czech Republic.*

Conklin, D. (2013b). Multiple Viewpoint Systems for Music Classification. *Journal of New Music Research*, *42*(1), 19–26.

Conklin, D., & Witten, I. H. (1995). Multiple Viewpoint Systems for Music Prediction. *Journal of New Music Research*, *24*(1), 51–73.

Cook, N. (1994). *A guide to Musical Analysis.* Oxford University Press, USA.

Cover, T., & Hart, P. (1967). Nearest Neighbor Pattern Classification. *IEEE transactions on Information Theory*, *13*(1), 21–27.

CPDL organisation. (2018). *Choral Public Domain Library.* Retrieved from `http://www2.cpdl.org/` (Accessed: 20-09-2018)

Cuthbert, M., & Ariza, C. (2010, 01). Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *ISMIR* (p. 637-642).

De Haas, W. B., Magalhães, J. P., Wiering, F., & C. Veltkamp, R. (2013). Automatic functional harmonic analysis. *Computer Music Journal*, *37*(4), 37–53.

De Haas, W. B., Magalhaes, J. P., Wiering, F., & Veltkamp, R. C. (2011). *Harmtrace: Automatic functional harmonic analysis* (Tech. Rep.). Technical Report UU-CS-2011-023, Department of Information and Computing ....

De Haas, W. B., Rohrmeier, M., Veltkamp, R. C., & Wiering, F. (2009). Modeling harmonic similarity using a generative grammar of tonal harmony. In *Proceedings of the tenth international conference on music information retrieval (ismir).*

Deutsch, L. P. (1996, May). *GZIP file format specification version 4.3* (No. 1952). RFC 1952. RFC Editor. Retrieved from `https://rfc-editor.org/rfc/rfc1952.txt` (Accessed: 23-09-2018)

Dijkstra, E. W. e. a. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Dobszay, L. (1972). The kodály method and its musical basis. *Studia Musicologica*, 15–33.

Downie, J. S. (2003). Music information retrieval. *Annual review of information science and technology*, *37*(1), 295–340.

Forte, A. (1974). *Tonal harmony in concept and practice*. Holt, Rinehart and Winston.

Frankland, B. W., & Cohen, A. J. (2004). Parsing of melody: Quantification and testing of the local grouping rules of Lerdahl and Jackendoff's A Generative Theory of Tonal Music. *Music Perception*, *21*(4), 499–543.

Gallé, M. (2011). *Searching for Compact Hierarchical Structures in DNA by means of the Smallest Grammar Problem* (Unpublished doctoral dissertation). Université Rennes 1.

Giraud, M., & Staworko, S. (2015). Modeling Musical Structure with Parametric Grammars. In *International Conference on Mathematics and Computation in Music* (pp. 85–96).

Godlovitch, S. (2002). *Musical performance: A philosophical study*. Routledge.

Goienetxea, I., Neubarth, K., & Conklin, D. (2016). Melody Classification with Pattern Covering. In *9th International Workshop on Music and Machine Learning (MML 2016), Riva del Garda, Italy.*

Good, M. (2001). MusicXML: An Internet-Friendly Format for Sheet Music. In *XML Conference and Expo.* (pp. 3–4).

Granroth-Wilding, M., & Steedman, M. (2014). A robust parser-interpreter for jazz chord sequences. *Journal of New Music Research*, *43*(4), 355–374.

Granroth-Wilding, M. T. (2013). Harmonic Analysis of Music using Combinatory Categorial Grammar.

Gritten, A., & King, E. (2006). *Music and Gesture.* Ashgate Publishing, Ltd.

Grumbach, S., & Tahi, F. (1994). A new challenge for compression algorithms: genetic sequences. *Information Processing & Management*, *30*(6), 875–886.

Hall, J. R. C., Hall, J. R., Battani, M., & Neitz, M. J. (2003). *Sociology on culture.* Psychology Press.

Hamanaka, M., Hirata, K., & Tojo, S. (2016). Implementing methods for analysing music based on lerdahl and jackendoff's generative theory of tonal music. In *Computational music analysis* (pp. 221–249). Springer.

Hamanaka, M., & Tojo, S. (2009). Interactive gttm analyzer. In *Ismir* (pp. 291–296).

Hillewaere, R., Manderick, B., & Conklin, D. (2014). Alignment Methods for Folk Tune Classification. In *Data Analysis, Machine Learning and Knowledge Discovery* (pp. 369–377). Springer.

Howard, D. M. (1952). *The oxford dictionary of nursery rhymes.* JSTOR.

Johannes Kepler University. (2013). *JKU Patterns Development Database (August 2013 version).* Retrieved from http://tomcollinsresearch.net/research/data/mirex/JKUPDD-Aug2013.zip (Accessed: 25-09-2018)

Kaser, V. A. (1993). Musical expressions of subconscious feelings: A clinical perspective. *Music Therapy Perspectives.*

Keiler, A. (1978). Bernstein's" the unanswered question" and the problem of musical competence. *The Musical Quarterly*, *64*(2), 195–222.

Kempf, D. (1996). What is symmetry in music? *International Review of the Aesthetics and Sociology of Music*, 155–165.

Kieffer, J. C., & Yang, E.-H. (2000). Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, *46*(3), 737–754.

Kohavi, R. e. a. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI* (Vol. 14, pp. 1137–1145).

Kolmogorov, A. N. (1963). On Tables of Random Numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, 369–376.

Lai, Y.-R., & Su, A. W.-Y. (2021). Deep learning based detection of gpr6 gttm global feature rule of music scores. In *Proceedings of the 8th international conference on new music concepts.*

Langley, P. (1995). Simplicity and representation change in grammar induction.

Lehman, E., & Shelat, A. (2002). Approximation Algorithms for Grammar-based Compression. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 205–212).

Lerch, A., Arthur, C., Pati, A., & Gururani, S. (2019). Music Performance Analysis: A Survey. *arXiv preprint arXiv:1907.00178*.

Lerdahl, F., & Jackendoff, R. (1983). *A Generative Theory of Tonal Music.* Cambridge MA: MIT Press.

Levenshtein, V. I. e. a. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (Vol. 10, pp. 707–710).

Li, M., Chen, X., Li, X., Ma, B., & Vitányi, P. M. B. (2004). The Similarity Metric. *IEEE transactions on Information Theory*, *50*(12), 3250–3264.

Li, M., & Vitányi, P. e. a. (2008). *An introduction to Kolmogorov complexity and its applications* (Vol. 3). Springer.

Louboutin, C., & Meredith, D. (2016). Using General-purpose Compression Algorithms for Music Analysis. *Journal of New Music Research*, *45*(1), 1–16.

Marsden, A. (1992). Modelling the perception of musical voices: a case study in rule-based systems. *Computer Representations and Models in Music*, 239–263.

Marsden, A. (2010). Schenkerian analysis by computer: A proof of concept. *Journal of New Music Research*, *39*(3), 269–289.

Marsden, A. (2012). Interrogating melodic similarity: a definitive phenomenon or the product of interpretation? *Journal of New Music Research*, *41*(4), 323–335.

Marsden, A. A. (2007). Automatic derivation of musical structure: A tool for research on schenkerian analysis. In *International conference on music information retrieval* (pp.

55–58).

McLeod, A., & Steedman, M. (2017). Meter Detection in Symbolic Music Using a Lexicalized PCFG. In *Proceedings of the 14th sound and music computing conference* (pp. 373–379).

McPherson, J. (2006). *Coordinating Knowledge to Improve Optical Music Recognition* (Unpublished doctoral dissertation). The University of Waikato.

Meertens Instituut. (2018). *The Meertens Tune Collections: MTC-ANN-2.0.1.* Retrieved from `http://www.liederenbank.nl/mtc/` (Accessed: 25-09-2018)

Meredith, D. (2006a). Point-Set Algorithms for Pattern Discovery and Pattern Matching in Music. In *Dagstuhl Seminar Proceedings.*

Meredith, D. (2006b). The ps13 Pitch Spelling Algorithm. *Journal of New Music Research, 35*(2), 121–159.

Meredith, D. (2014). Using Point-Set Compression to Classify Folk Songs. In *International Workshop on Folk Music Analysis* (pp. 29–35).

Meredith, D., Lemström, K., & Wiggins, G. A. (2002). Algorithms for discovering Repeated Patterns in Multidimensional Representations of Polyphonic Music. *Journal of New Music Research, 31*(4), 321–345.

Meredith, David. (2013). *COSIATEC, 2013 MIREX competition on Discovery of Repeated Themes and Sections submission.* Retrieved from `http://titanmusic.com/software/mirex2013/Mirex2013CosiatecRaw.zip` (Accessed: 23-09-2018)

MIDI Association. (1996). *MIDI 1.0 Detailed Specifications.* Retrieved from `https://www.midi.org/specifications-old/item/the-midi-1-0-specification` (Accessed: 28-09-2020)

Mondol, T. (2020). *Style recognition in music with context free grammars and kolmogorov complexity* (Unpublished master's thesis). University of Waterloo.

Mondol, T., & Brown, D. G. (2021). Grammar-based compression and its use in symbolic music analysis. *Journal of Mathematics and Music*, 1–18.

Musopen organisation. (2018). *Musopen - Free sheet music, royalty free music, and public domain resources.* Retrieved from `https://musopen.org/` (Accessed: 20-09-2018)

Nápoles, N., Vigliensoni, G., & Fujinaga, I. (2018). Encoding matters. In *Proceedings of the 5th international conference on digital libraries for musicology* (pp. 69–73).

Navarro, G. (2001). A guided tour to approximate string matching. *ACM computing surveys (CSUR), 33*(1), 31–88.

Needleman, S. B., & Wunsch, C. D. (1970). A General Method applicable to the Search for Similarities in the Amino Acid Sequence of two Proteins. *Journal of Molecular Biology*, *48*(3), 443–453.

Nevill-Manning, C. G., & Witten, I. H. (1997). Identifying Hierarchical Structure in Sequences: A Linear-time Algorithm. *Journal of Artificial Intelligence Research*, *7*, 67–82.

Oppenheim, I., Walshaw, C., & Atchley, J. (2011). *The abc Standard 2.1.* Retrieved from `http://abcnotation.com/wiki/abc:standard:v2.1` (Accessed: 25-09-2018)

Palisca, C. V. (2000). Moving the affections through music: pre-cartesian psycho-physiological theories. In *Number to sound* (pp. 289–308). Springer.

Pearce, M. T. (2018). Statistical learning and probabilistic prediction in music cognition: mechanisms of stylistic enculturation. *Annals of the New York Academy of Sciences*, *1423*(1), 378–395.

Pearce, M. T., Müllensiefen, D., & Wiggins, G. A. (2008). A Comparison of Statistical and Rule-Based Models of Melodic Segmentation. In *ISMIR* (pp. 89–94).

Purwins, H., Grachten, M., Herrera, P., Hazan, A., Marxer, R., & Serra, X. (2008). Computational models of music perception and cognition ii: Domain-specific music processing. *Physics of Life Reviews*, *5*(3), 169–182.

Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica*, *14*(5), 465–471.

Roads, C., & Wieneke, P. (1979). Grammars as representations for music. *Computer Music Journal*, 48–55.

Rohrmeier, M. (2007). A generative grammar approach to diatonic harmonic structure. In *Proceedings of the 4th sound and music computing conference* (pp. 97–100).

Rohrmeier, M. (2011). Towards a generative syntax of tonal harmony. *Journal of Mathematics and Music*, *5*(1), 35–53.

Rohrmeier, M., & Cross, I. (2008). Statistical properties of tonal harmony in bach's chorales. In *Proceedings of the 10th international conference on music perception and cognition* (pp. 619–627).

Rytter, W. (2002). Application of Lempel-Ziv Factorization to the Approximation of Grammar-based Compression. In *Annual Symposium on Combinatorial Pattern Matching* (pp. 20–31).

Sabrebd. (2009). *The main melody of Three Blind Mice.* Published under Creative Commons Attribution-Share Alike 3.0 Unported license (https://creativecommons.org/licenses/by-sa/3.0/deed.en). Wikimedia Commons. Retrieved from `https://commons.wikimedia.org/wiki/File:3BlindMice.jpg` (Accessed: 24-01-2023)

Sachs, C. (1948). Some remarks about old notation. *The Musical Quarterly*, *34*(3), 365–370.

Sapp, C. S. (2005). Online Database of Scores in the Humdrum File Format. In *ISMIR* (pp. 664–665).

Savakis, A. E. (2000). Evaluation of Lossless Compression Methods for Gray Scale Document Images. In *Image Processing, 2000. Proceedings. 2000 International Conference on* (Vol. 1, pp. 136–139).

Schaffrath, H., & Huron, D. (1995). The Essen Folksong Collection in the Humdrum Kern Format. *Menlo Park, CA: Center for Computer Assisted Research in the Humanities*.

Schenker, H., & Salzer, F. (1969). *Five Graphic Music Analyses (F nf Urlinie-Tafeln)*. Courier Corporation.

Schoenberg, A., Stein, L., & Strang, G. (1967). *Fundamentals of Musical Composition*. Faber & Faber London.

Seward, J. (1996). *BZIP2 and LibBZIP2.* Retrieved from `http://www.bzip.org/` (Accessed: 23-09-2018)

Shanmugasundaram, S., & Lourdusamy, R. (2011). A Comparative Study of Text Compression Algorithms. *International Journal of Wisdom Based Computing*, *1*(3), 68–76.

Shimbel, A. (1954). Structure in communication nets. In *Proceedings of the symposium on information networks* (pp. 119–203).

Sidorov, K. A. (2015). *sib2ly: A converter of music from Sibelius to LilyPond.* Retrieved from `https://sourceforge.net/projects/sib2ly/` (Accessed: 21-09-2018)

Sidorov, K. A., Jones, A., & Marshall, A. D. (2014). Music Analysis as a Smallest Grammar Problem. In *ISMIR* (pp. 301–306).

Singer, J. (2004). Creating a Nested Melodic Representation: Competition and Cooperation among Bottom-up and Top-down Gestalt Principles. In *ISMIR.*

Siyari, P., & Gallé, M. (2017). The Generalized Smallest Grammar Problem. In *International Conference on Grammatical Inference* (pp. 79–92).

Sleator, Daniel and Temperley, David. (1999). *The Melisma Music Analyzer.* Retrieved from `https://www.link.cs.cmu.edu/music-analysis/` (Accessed: 08-02-2023)

Smith, A. (1999). Why digitize?

Sober, E. (2015). *Ockham's Razors*. Cambridge University Press.

Solomonoff, R. J. (1964). A formal theory of inductive inference. part i. *Information and control*, *7*(1), 1–22.

Spring, G., & Hutcheson, J. (2013). *Musical form and analysis: Time, pattern, proportion*. Waveland Press.

Steedman, M., & Baldridge, J. (2011). Combinatory categorial grammar. *Non-Transformational Syntax: Formal and explicit models of grammar*, 181–224.

Steedman, M. J. (1984). A Generative Grammar for Jazz Chord Sequences. *Music Perception: An Interdisciplinary Journal*, *2*(1), 52–77.

Stober, S. (2011). Adaptive Distance Measures for Exploration and Structuring of Music Collections. In *Audio Engineering Society Conference: 42nd International Conference: Semantic Audio.*

Stoye, J., & Gusfield, D. (2002). Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science*, *270*(1-2), 843–856.

Student. (1908). The Probable Error of a Mean. *Biometrika*, 1–25.

Stutter, J. J. (2020). *The musical, notational and codicological evidence of W1 for an oral transmission of Notre Dame polyphony to Scotland* (Unpublished doctoral dissertation). University of Glasgow.

Stützle, T., & Hoos, H. H. (2000). Max–min ant system. *Future generation computer systems*, *16*(8), 889–914.

Sundberg, J., & Lindblom, B. (1976). Generative theories in language and music descriptions. *Cognition*, *4*(1), 99–122.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. MIT Press.

Temperley, D. (2007). *Music and probability*. MIT Press.

Thulasiraman, K., & Swamy, M. N. (2011). *Graphs: theory and algorithms*. John Wiley & Sons.

Tsukiyama, T., Kondo, Y., Kakuse, K., Saba, S., Ozaki, S., & Itoh, K. (1986, April 29). *Method and System for Data Compression and Restoration.* Google Patents. (US Patent 4,586,027)

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, *58*(345-363), 5.

Tymoczko, D. (2003). Function theories: A statistical approach. *Musurgia*, *10*(3-4), 35–64.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, *14*(3), 249–260.

Ulrich, J. W. (1977). The analysis and synthesis of jazz by computer. In *Ijcai* (pp. 865–872).

van Kranenburg, P., Janssen, B., & Volk, A. (2016). *The Meertens Tune Collections: The Annotated Corpus (MTC-ANN) versions 1.1 and 2.0.1* (Vol. 2016) (No. 1). Meertens instituut KNAW.

van Kranenburg, P., Volk, A., & Wiering, F. (2013). A Comparison between Global and Local Features for Computational Classification of Folk Song Melodies. *Journal of New Music Research*, *42*(1), 1–18.

Vanlehn, K., & Ball, W. (1987). A version space approach to learning context-free grammars. *Machine learning*, *2*(1), 39–74.

Velarde, G., Weyde, T., & Meredith, D. (2013). An Approach to Melodic Segmentation and Classification based on Filtering with the Haar-Wavelet. *Journal of New Music Research*, *42*(4), 325–345.

Vitányi, P. M. B., & Li, M. (2000). Minimum Description Length Induction, Bayesianism, and Kolmogorov Complexity. *IEEE Transactions on Information Theory*, *46*(2), 446–464.

Volk, A., & Van Kranenburg, P. (2012). Melodic Similarity among Folk Songs: An Annotation Study on Similarity-based Categorization in Music. *Musicae Scientiae*, *16*(3), 317–339.

Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, *21*(1), 168–173.

Watson, M. (2018). Musescore. *Journal of the Musical Arts in Africa*, *15*(1-2), 143–147.

Welch, T. A. (1984). Technique for High-Performance Data Compression. *Computer*, *52*.

Winograd, T. (1968). Linguistics and the computer analysis of tonal harmony. *journal of Music Theory*, *12*(1), 2–49.

Ziv, J., & Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, *23*(3), 337–343.

Ziv, J., & Lempel, A. (1978). Compression of Individual Sequences via Variable-Rate Coding. *IEEE transactions on Information Theory*, *24*(5), 530–536.