MUSIC ANALYSIS AS A SMALLEST GRAMMAR PROBLEM

Kirill Sidorov **Andrew Jones** **David Marshall**
Cardiff University, UK
{K.Sidorov, Andrew.C.Jones, Dave.Marshall}@cs.cardiff.ac.uk

ABSTRACT

In this paper we present a novel approach to music analysis, in which a grammar is automatically generated explaining a musical work's structure. The proposed method is predicated on the hypothesis that the shortest possible grammar provides a model of the musical structure which is a good representation of the composer's intent. The effectiveness of our approach is demonstrated by comparison of the results with previously-published expert analysis; our automated approach produces results comparable to human annotation. We also illustrate the power of our approach by showing that it is able to locate errors in scores, such as introduced by OMR or human transcription. Further, our approach provides a novel mechanism for intuitive high-level editing and creative transformation of music. A wide range of other possible applications exists, including automatic summarization and simplification; estimation of musical complexity and similarity, and plagiarism detection.

1. INTRODUCTION

In his Norton Lectures [1], Bernstein argues that music can be analysed in linguistic terms, and even that there might be "a worldwide, inborn musical grammar". Less specifically, the prevalence of musical form analyses, both large-scale (*e.g.* sonata form) and at the level of individual phrases, demonstrates that patterns, motifs, *etc.,* are an important facet of a musical composition, and a grammar is certainly one way of capturing these artefacts.

In this paper we present a method for automatically deriving a compact grammar from a musical work and demonstrate its effectiveness as a tool for analysing musical structure. A key novelty of this method is that it operates *automatically*, yet generates insightful results. We concentrate in this paper on substantiating our claim that generating parsimonious grammars is a useful analysis tool, but also suggest a wide range of scenarios to which this approach could be applied.

Previous research into grammar-based approaches to modelling music has led to promising results. Treating harmonic phenomena as being induced by a generative grammar has been proposed in [9, 24, 27], and the explanatory power of such grammars has been demonstrated. The use of musical grammar based of the Generative Theory of Tonal Music [19] has been proposed in [11–13], for the analysis of music. Similarly, there is a number of grammar-based approaches to automatic composition, including some which automatically learn stochastic grammars or derive grammars in an evolutionary manner, although some researchers continue to craft grammars for this purpose by hand [7].

However, in these works the derivation of grammar rules themselves is performed manually [27] or semi-automatically [11–13] from heuristic musicological considerations. In some cases generative grammars (including stochastic ones) are derived or learned automatically, but they describe general patterns in a corpus of music, *e.g.* for synthesis [16, 22], rather than being precise analyses of individual works. In a paper describing research carried out with a different, more precise aim of visualising semantic structure of an individual work, the authors remark that they resorted to manual retrieval of musical structure data from descriptive essays "since presently there is no existing algorithm to parse the high-level structural information automatically from MIDI files or raw sound data" [5].

In this paper we present a method which addresses the above concern expressed by Chan *et al.* in [5], but which at the same time takes a principled, information-theoretical approach. We argue that the best model explaining a given piece of music is the most compact one. This is known as Minimum Description Length principle [23] which, in turn, is a formal manifestation of the Occam's razor principle: the best explanation for data is the most compressive one. Hence, given a piece of music, we seek to find the shortest possible context free grammar that generates this piece (and only this piece). The validity of our compressive modelling approach in this particular domain is corroborated by evidence from earlier research in predictive modelling of music [6] and from perception psychology [14, 25]: humans appear to find strongly compressible music (which therefore has a compact grammar) appealing.

2. COMPUTING THE SMALLEST GRAMMAR

Given a piece of music, we treat it as a sequence(s) of symbols (see Section 2.1) and we seek to find the *shortest possible* context-free grammar that generates this (and only this) piece. Following [21], we define the size of a grammar G to be the total length of the right hand sides of all the production rules R_i plus one for each rule (length of a separator or cost of introducing a new rule):

$$|G| = \sum_i (|R_i| + 1)\,. \qquad (1)$$

Searching for such a grammar is known as the *smallest grammar problem*. It has recently received much attention due its importance in compression and analysis of DNA sequences, see *e.g.* [4]. For an overview of the smallest grammar problem the reader is referred to [3].

Computing the smallest grammar is provably NP-hard (see [18] Theorem 3.1), therefore in practice we are seeking an approximation to the smallest grammar.

Various heuristics have been proposed in order to tackle the smallest grammar problem in tractable time by greedy algorithms [4, 21]. A fast on-line (linear time) algorithm called SEQUITUR has been proposed in [20]. While the focus of [20] was fast grammar inference for large sequences, rather than strong compression, [20] contains an early mention that such techniques may be applied to parsing of music. (In Section 3 we compare grammars produced by SEQUITUR with our approach.)

In [4, 21], a class of algorithms involving iterative replacement of a repeated substring is considered (termed there iterative repeat replacement (IRR)). We employ a similar procedure here, summarised in Alg. 1. First, the grammar G is initialised with top level rule(s), whose right-hand sides initially are simply the input string(s). Then, in the original IRR scheme, a candidate substring c is selected according to some scoring function F. All non-overlapping occurrences of this substring in the grammar are replaced with a new symbol R_{n+1}, and a new rule is added to the grammar: $R_{n+1} \rightarrow c$. Replacement of a substring of length m in a string of length n can be done using the Knuth-Morris-Pratt algorithm [17] in $O(m+n)$ time. The replacement procedure repeats until no further improvement is possible.

In [4, 21] the various heuristics according to which such candidate substitutions can be selected are examined; the conclusion is that the "locally most compressive" heuristic results in the shortest final grammars. Suppose a substring of length L occurring N times is considered for replacement with a new rule. The resulting saving is, therefore [21]:

$$F = \Delta|G| = (LN) - (L + 1 + N). \qquad (2)$$

Hence, we use Eq. (2) (the locally most compressive heuristic) as our scoring function when selecting candidate substrings (in line 2 of Alg. 1). We found that the greedy iterative replacement scheme of [21] does not always produce optimal grammars. We note that a small decrease in grammar size may amount to a substantial change in the grammar's *structure*, therefore we seek to improve the compression performance.

To do so, instead of greedily making a choice at each iteration, we recursively evaluate (line 9) multiple (w) candidate substitutions (line 2) with backtracking, up to a certain depth $d_{\max}$ (lines 9–15). Once the budgeted search depth has been exhausted, the remaining substitutions are done greedily (lines 4–7) as in [21]. This allows us to control the greediness of the algorithm from completely greedy ($d_{\max} = 0$) to exhaustive search ($d_{\max} = \infty$). We observed that using more than 2–3 levels of backtracking usually does not yield any further reduction in the size of the grammar.

Algorithm 1 `CompGram` (Compress grammar)

```
Require: Grammar $G$; search depth $d$.
   (Tuning constants: max depth $d_{\max}$; width $w$)
 1: loop
 2:   Find $w$ best candidate substitutions $C = \{c_i\}$ in $G$.
 3:   if $C = \varnothing$ then return $G$; end if
 4:   if recursion depth $d > d_{\max}$ then
 5:     Greedily choose best $c_{\text{best}}$
 6:     $G' :=$ replace($G$, $c_{\text{best}}$, new symbol)
 7:     return CompGram($G'$, $d+1$)
 8:   else
 9:     Evaluate candidates:
10:     for $c_i \in C$ do
11:       $G' :=$ replace($G$, $c_i$, new symbol)
12:       $G''_i :=$ CompGram($G'$, $d+1$)
13:     end for
14:     $b := \arg\min_i |G''_i|$
15:     return $G''_b$
16:   end if
17: end loop
```

Selecting a candidate according to Eq. (2) in line 2 involves maximising the number of non-overlapping occurrences of a substring, which is known as the *string statistics problem*, the solutions to which are not cheap [2]. Therefore, as in [4] we approximate the maximal number of non-overlapping occurrences with the number of *maximal repeats* [10]. All z maximal repeats in a string (or a set of strings) of total length n can be found very fast (in $O(n + z)$ time) using suffix arrays [10]. In principle, it is possible to construct an example in which this number will be drastically different from the true number of non-overlapping occurrences (*e.g.* a long string consisting of a repeated symbol). However, this approximation was shown to work well in [4] and we have confirmed this in our experiments. Further, this concern is alleviated by the backtracking procedure we employ.

2.1 Representation of Music

In this paper, we focus on music that can be represented as several monophonic voices (such as voices in a fugue, or orchestral parts), that is, on the horizontal aspects of the music. We treat each voice, or orchestral part, as a string. We use (diatonic) intervals between adjacent notes, ignoring rests, as symbols in our strings. For ease of explanation of our algorithm we concentrate on the melodic information only, ignoring rhythm (note durations). Rhythmic invariance may be advantageous when melodic analysis is the prime concern. However, it is trivial to include note durations, and potentially even chord symbols and other musical elements, as symbols in additional (top level) strings.

Note that even though we take no special measures to model the relationship between the individual voices, this is happening *automatically*: indeed, all voices are encompassed in the same grammar and are considered for the iterative replacement procedure on equal rights as the grammar is updated.

Rule	R_9	R_8	R_5	R_{18}	R_{10}	R_{11}	R_4	R_{19}	R_7	R_{20}
Freq.	2	2	4	2	5	5	2	2	3	3
Len.	16	12	6	7	5	5	4	6	5	4
E. len.	99	94	24	47	11	11	37	34	17	16
Comp.	**96**	**91**	**67**	**44**	**38**	**38**	**34**	**31**	**30**	**28**

Table 1. Grammar statistics for Fugue №10. The ten *most compressing* rules are shown. For each rule R_i: *Freq.* is the number of times a rule occurs in the grammar, *Len.* is its right hand side length, *E. len.* is the length of the rule's expansion, and *Comp.* is the total saving due to this rule.

3. RESULTS AND APPLICATIONS

3.1 Automatic Structural Analysis

We have applied our method to automatically detect the structure of a selection of Bach's fugues. (Eventually we intend to analyse all of them in this way.) Figure 1 shows one example of such analysis. We show the voices of the fugue in piano roll representation, with the hierarchy of the grammar on top: rules are represented by brackets labelled by rule number. For completeness, we give the entire grammar (of size $|G| = 217$) obtained for Fugue №10 later, in Fig. 7. Figure 2 zooms in onto a fragment of the score with the rules overlaid. For comparison, we show manual analysis by a musicologist [26] in Fig. 3. Observe that all the main structural elements of the fugue have been correctly identified by our method (*e.g.* exp. and 1st dev. in rule R_8, re-exp. and 4th dev. in rule R_9, variant of re-exp. and 2nd dev in R_{18} and R_{19}) and our automatic analysis is comparable to that by a human expert.

It is possible to use structures other than individual notes or intervals as symbols when constructing grammars. Figure 4 shows the simplified grammar for Fugue №10 generated using entire bars as symbols. In this experiment we first measured pairwise similarity between all bars (using Levenshtein distance [8]) and denoted each bar by a symbol, with identical or almost identical bars being denoted by the same symbol. The resulting grammar (Fig. 4) can be viewed as a coarse-grained analysis. Observe again that it closely matches human annotation (Fig. 3).

Our approach can also be used to detect prominent high-level features in music. We can compute the usage frequency for each rule and the corresponding savings in grammar size (as shown in Table 1 for Fugue №10). Most compressing rules, we argue, correspond to structurally important melodic elements. The present example illustrates our claim: rule R_8 corresponds to the fugue's exposition, R_9 to re-exposition, and R_5 to the characteristic chromatic figure in the opening (*cf.* Figs. 1 to 3 and the score).

In addition to high-level analysis, our approach can be used to detect the smallest constituent building blocks of a piece. For example, Fig. 5 shows the lowest level rules (that use only terminals) produced in analysis of Fugue №10, and the frequency of each rule. These are the elementary "bricks" from which Bach has constructed this fugue.

In [20], SEQUITUR is applied to two Bach chorales. In Fig. 6 we replicate the experiment from [20] and com-

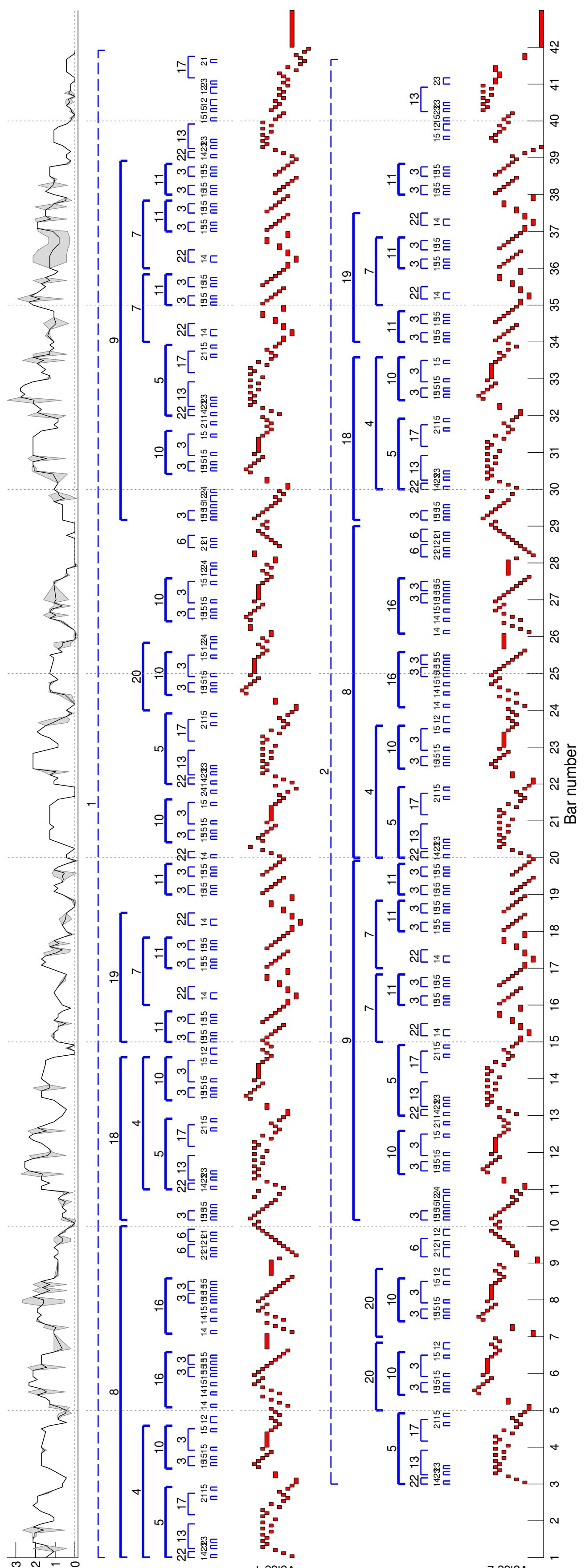

Figure 1. Automatic analysis of Bach's Fugue №10 from WTK book I. *On top*: sensitivity to point errors as measured by the increase in grammar size Δ|G|.

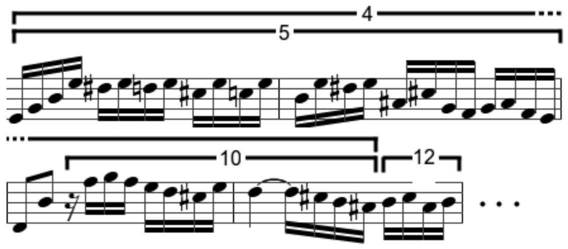

Figure 2. Close-up view of the first four bars: rules R_4, R_5, R_{10}, and R_{12} overlaid with the score (lower level rules are not shown).

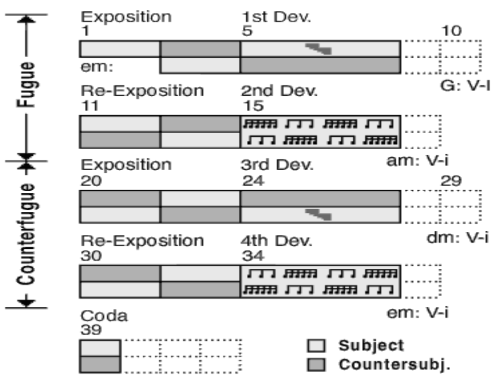

Figure 3. Manual analysis of Fugue №10 by a musicologist (from [26] with permission).

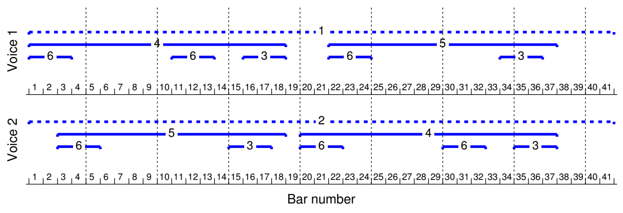

Figure 4. Simplified automatic analysis of Fugue №10 using whole bars as symbols.

pare the grammars of these two chorales generated by SEQUITUR and by our approach. The chorales are very similar except for a few subtle differences. We note that our method was able to produce a shorter grammar ($|G_{\text{our}}| = 50$ vs. $|G_{\text{Sequitur}}| = 59$) and hence revealed more of the relevant structure, while the grammar of the more greedy (and hence less compressing) SEQUITUR was compromised by the small differences between the chorales.

3.2 Error Detection and Spell-checking

We investigated the sensitivity of the grammars generated by our method to alterations in the original music. In one experiment, we systematically altered each note in turn (introducing a point error) in the 1st voice of Fugue №10 and constructed a grammar for each altered score. The change in grammar size relative to that of the unaltered score is plotted in Fig. 1 (*top*) as a function of the alteration's position. Observe that the grammar is more sensitive to alterations in structurally dense regions and less sensitive elsewhere, *e.g.* in between episodes. Remarkably, with

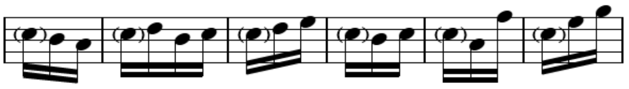

Figure 5. Atomic (lowest level) rules with their frequencies in brackets.

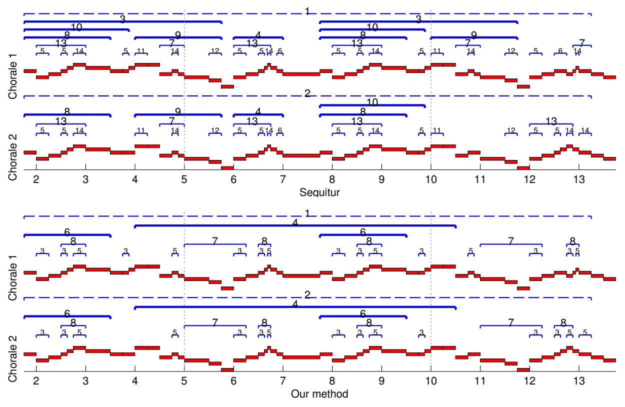

Figure 6. The grammars for the Bach chorales (from [20]) produced by SEQUITUR (*above*), $|G_{\text{Sequitur}}| = 59$, and by the proposed approach (*below*), $|G_{\text{our}}| = 50$.

very few exceptions (*e.g.* bars 10, 26) altering the piece *consistently* results in the grammar size *increasing*. We observed a similar effect in other Bach fugues and even in 19th century works (see below). We propose, only partially in jest, that this indicates that Bach's fugues are close to structural perfection which is ruined by even the smallest alteration.

Having observed the sensitivity of the grammar size to point errors (at least in highly structured music), we propose that grammar-based modelling can be used for musical "spell-checking" to correct errors in typesetting (much like a word processor does for text), or in optical music recognition (OMR). This is analogous to compressive sensing which is often used in signal and image processing (see *e.g.* [15]) for denoising: noise compresses poorly. We can regard errors in music as noise and use the grammar-based model for locating such errors. We investigated this possibility with the following experiment.

As above, we introduce a point error (replacing one note) at a random location in the score to simulate a "typo" or an OMR error. We then systematically alter every note in the score and measure the resulting grammar size in each case. When the error is thus undone by one of the modifications, the corresponding grammar size should be noticeably smaller, and hence the location of the error may thus be revealed. We rank the candidate error positions by grammar size and consider suspected error locations with grammar size less than or equal to that of the ground truth error, *i.e.* the number of locations that would need to be manually examined to pin-point the error, as false positives. We then report the number of false positives as a fraction of the total number of notes in the piece. We repeat the experiment for multiple randomly chosen error

$\boldsymbol{S}_1 \rightarrow R_8\ 2{\uparrow}\ R_{18}\ R_{12}\ 1{\uparrow}\ R_{19}\ 5{\downarrow}\ 7{\uparrow}\ R_{11}\ 1{\uparrow}\ R_{22}\ 3{\downarrow}\ R_{10}\ 2{\downarrow}\ R_{24}\ 4{\downarrow}\ R_5\ R_{20}\ R_{24}\ 0\ R_{10}\ R_{12}\ R_{24}\ 6{\downarrow}\ R_6\ 2{\downarrow}\ 1{\uparrow}\ 1{\downarrow}\ 3{\uparrow}\ R_9\ R_{22}\ R_{13}\ R_{15}\ 3{\uparrow}\ R_{15}\ R_{12}\ 1{\downarrow}\ R_{12}\ R_{23}\ R_{17}\ 1{\downarrow}\ 4{\uparrow}$

$\boldsymbol{S}_2 \rightarrow R_5\ R_{20}\ 9{\downarrow}\ R_{20}\ 9{\downarrow}\ 5{\uparrow}\ R_6\ R_{12}\ 1{\downarrow}\ 2{\uparrow}\ R_9\ R_8\ 3{\uparrow}\ R_{18}\ 2{\downarrow}\ 2{\downarrow}\ 2{\uparrow}\ 3{\uparrow}\ R_{19}\ 7{\downarrow}\ 9{\uparrow}\ R_{11}\ 1{\uparrow}\ 3{\downarrow}\ 2{\downarrow}\ 2{\downarrow}\ 11{\uparrow}\ 1{\uparrow}\ R_{15}\ 1{\downarrow}\ R_{12}\ R_{15}\ 7{\uparrow}\ R_{13}\ R_{23}\ 7{\downarrow}\ 4{\downarrow}$

$\boldsymbol{R}_3 \rightarrow R_{15}\ R_{15}$ $\quad$ $\boldsymbol{R}_4 \rightarrow R_5\ 5{\uparrow}\ 4{\uparrow}\ R_{10}$
$\boldsymbol{R}_5 \rightarrow R_{22}\ R_{13}\ 3{\uparrow}\ 1{\downarrow}\ R_{17}\ R_{15}$ $\quad$ $\boldsymbol{R}_6 \rightarrow R_{21}\ R_{21}\ 1{\uparrow}$
$\boldsymbol{R}_7 \rightarrow 2{\downarrow}\ R_{22}\ 5{\downarrow}\ 5{\uparrow}\ R_{11}$
$\boldsymbol{R}_8 \rightarrow R_4\ R_{12}\ 1{\uparrow}\ 4{\downarrow}\ R_{16}\ 6{\downarrow}\ R_{16}\ 2{\downarrow}\ 4{\downarrow}\ R_6\ R_6\ 1{\downarrow}$
$\boldsymbol{R}_9 \rightarrow R_3\ R_{15}\ R_{12}\ R_{24}\ 4{\uparrow}\ R_{10}\ 1{\downarrow}\ R_{21}\ 4{\downarrow}\ R_5\ R_7\ 1{\downarrow}\ R_7\ 3{\uparrow}\ R_{11}\ 1{\uparrow}$
$\boldsymbol{R}_{10} \rightarrow 1{\uparrow}\ R_3\ 2{\uparrow}\ R_3\ 1{\uparrow}$
$\boldsymbol{R}_{11} \rightarrow R_3\ 1{\downarrow}\ 5{\uparrow}\ R_3\ 1{\downarrow}$ $\quad$ $\boldsymbol{R}_{12} \rightarrow 1{\uparrow}\ 2{\downarrow}\ 1{\uparrow}$
$\boldsymbol{R}_{13} \rightarrow R_{23}\ R_{23}\ 2{\downarrow}\ 2{\uparrow}\ 2{\downarrow}\ 2{\uparrow}\ 3{\downarrow}$ $\quad$ $\boldsymbol{R}_{14} \rightarrow 2{\uparrow}\ 2{\uparrow}$
$\boldsymbol{R}_{15} \rightarrow 1{\downarrow}\ 1{\downarrow}$
$\boldsymbol{R}_{16} \rightarrow R_{14}\ 2{\uparrow}\ 4{\downarrow}\ R_{14}\ 2{\uparrow}\ R_{15}\ 2{\uparrow}\ R_3\ R_3\ 5{\uparrow}$
$\boldsymbol{R}_{17} \rightarrow 1{\uparrow}\ 4{\downarrow}\ 2{\uparrow}\ 3{\downarrow}\ 1{\downarrow}\ R_{21}\ 2{\downarrow}$
$\boldsymbol{R}_{18} \rightarrow R_3\ R_{15}\ 5{\uparrow}\ 6{\downarrow}\ 5{\uparrow}\ 6{\downarrow}\ R_4$
$\boldsymbol{R}_{19} \rightarrow R_{11}\ 1{\downarrow}\ R_7\ 1{\downarrow}\ 2{\downarrow}\ R_{22}$ $\quad$ $\boldsymbol{R}_{20} \rightarrow 5{\uparrow}\ 7{\uparrow}\ R_{10}\ R_{12}$
$\boldsymbol{R}_{21} \rightarrow 1{\uparrow}\ 1{\uparrow}$ $\quad$ $\boldsymbol{R}_{22} \rightarrow R_{14}\ 3{\uparrow}$
$\boldsymbol{R}_{23} \rightarrow 1{\downarrow}\ 1{\uparrow}$ $\quad$ $\boldsymbol{R}_{24} \rightarrow 2{\downarrow}\ 5{\uparrow}$

Figure 7. Automatically generated shortest grammar for Fugue №10. Here, $\boldsymbol{R}_i$ are production rules ($\boldsymbol{S}_i$ are the top level rules corresponding to entire voices), numbers with arrows are terminal symbols (diatonic intervals with the arrows indicating the direction).

Piece	F/P	Piece	F/P
Fugue №1[1]	6.07%	Fugue №2	2.28%
Fugue №10	1.55%	Fugue №9	9.07%
Bvn. 5th str.	2.28%	Elgar Qrt.	14.22%
Blz. SF. str.	16.71%	Mndsn. Heb.	16.28%

[1]Fugues are from WTC book I.

Table 2. Spell-checking performance: fraction of false positives.

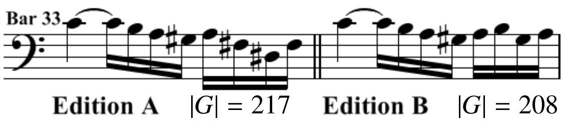

Figure 8. Selecting between two editions of Fugue №10 using grammar size.

locations and report median performance over 100 experiments in Table 2. We have performed this experiment on Bach fugues, romantic symphonic works (Beethoven's 5th symphony 1st mvt., Berlioz's "Symphonie Fantastique" 1st mvt., Mendelssohn's "Hebrides") and Elgar's Quartet 3rd mvt. We observed impressive performance (Table 2) on Bach's fugues (error location narrowed down to just a *few percent* of the score's size), and even in supposedly less-structured symphonic works the algorithm was able to *substantially narrow down* the location of potential errors. This suggests that our approach can be used to effectively locate errors in music: for example a notation editor using our method may highlight potential error locations, thus warning the user, much like word processors do for text.

A variant of the above experiment is presented in Fig. 8. We want to select between two editions of Fugue №10 in which bar 33 differs. We measured the total grammar size for the two editions and concluded that the variant in Edition B is more logical as it results in smaller grammar size $|G| = 208$ (vs. $|G| = 217$ for Edition A).

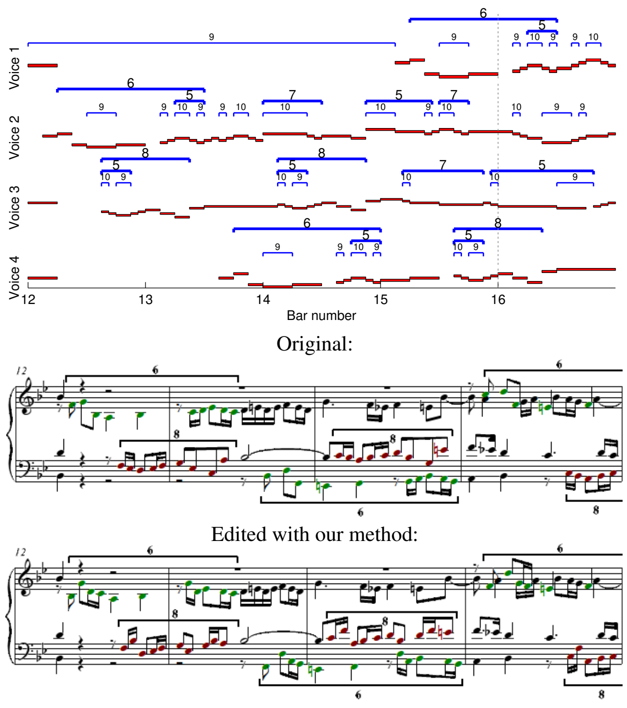

Figure 9. High-level editing. *Above*: automatic analysis of Fugue №16 (fragment); *middle*: original; *below*: rules R_6 and R_8 were edited with our method to obtain a new fugue.

3.3 High-level Editing

A grammar automatically constructed for a piece of music can be used as a means for high-level editing. For example, one may edit the right-hand sides of individual rules to produce a new similarly-structured piece, or, by operating on the grammar tree, alter the structure of a whole piece. We illustrate such editing in Fig. 9. We have automatically analysed Fugue №16 with our method and then edited two of the detected rules (R_6 and R_8) to obtain a new fugue (expanding the grammar back). This new fugue is partially based on new material, yet maintains the structural perfection of the original fugue. We believe this may be a useful and intuitive next-generation method for creatively transforming scores.

3.4 Further Applications

We speculate that in addition to the applications discussed above, the power of our model may be used in other ways: for estimation of complexity and information content in a musical piece; as means for automatic summarisation by analysing the most compressive rules; for improved detection of similarity and plagiarism (including structural similarity); for automatic simplification of music (by transforming a piece so as to decrease its grammar size); and for classification of music according to its structural properties.

Having observed that size of grammar is a good measure of the "amount of structure" in a piece, we suggest that our model can even be used to tell good music from bad music.

Hypothetically, a poor composition would remain poor even when (random) alterations are made to it and hence its grammar size would be insensitive to such alterations, while well-constructed works (like those of Bach in our examples) would suffer, in terms of grammar size, from perturbation.

4. CONCLUSIONS AND FUTURE WORK

We have posed the analysis of music as a smallest grammar problem and have demonstrated that building parsimonious context-free grammars is an appealing tool for analysis of music, as grammars give insights into the underlying structure of a piece. We have discussed how such grammars may be efficiently constructed and have illustrated the power of our model with a number of applications: automatic structural analysis, error detection and spell-checking (without prior models), high-level editing.

Future work would include augmenting the presented automatic grammatical analysis to allow inexact repetitions (variations or transformations of material) to be recognised in the grammar, and, in general, increasing the modelling power by recognising more disguised similarities in music.

5. REFERENCES

[1] L. Bernstein. Norton lectures. `http://www.leonardbernstein.com/norton.htm`.

[2] G. Brodal, R. B. Lyngsø, Anna Östlin, and Christian N. S. Pedersen. Solving the string statistics problem in time $O(n \log n)$. In *Proc. ICALP '02*, pages 728–739, 2002.

[3] R. Carrascosa, F. Coste, M. Gallé, and G. G. I. López. The smallest grammar problem as constituents choice and minimal grammar parsing. *Algorithms*, 4(4):262–284, 2011.

[4] R. Carrascosa, F. Coste, M. Gallé, and G. G. I. López. Searching for smallest grammars on large sequences and application to DNA. *J. Disc. Alg.*, 11:62–72, 2012.

[5] W.-Y. Chan, H. Qu, and W.-H. Mak. Visualizing the semantic structure in classical music works. *IEEE Trans. Vis. and Comp. Graph.*, 16(1):161–173, 2010.

[6] D. Conklin and I. H. Witten. Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24:51–73, 1995.

[7] J. D. Fernandez and F. J. Vico. AI methods in algorithmic composition: A comprehensive survey. *J. Artif. Intell. Res. (JAIR)*, 48:513–582, 2013.

[8] M. Grachten, J. L. Arcos, and R. L. de Mántaras. Melodic similarity: Looking for a good abstraction level. In *Proc. ISMIR*, 2004.

[9] M. Granroth-Wilding and M. Steedman. Statistical parsing for harmonic analysis of jazz chord sequences. In *Proc. ICMC*, pages 478–485, 2012.

[10] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[11] M. Hamanaka, K. Hirata, and S. Tojo. Implementing a generative theory of tonal music. *Journal of New Music Research*, 35(4):249–277, 2006.

[12] M. Hamanaka, K. Hirata, and S. Tojo. FATTA: Full automatic time-span tree analyzer. In *Proc. ICMC*, pages 153–156, 2007.

[13] M. Hamanaka, K. Hirata, and S. Tojo. Grouping structure generator based on music theory GTTM. *J. of Inf. Proc. Soc. of Japan*, 48(1):284–299, 2007.

[14] N. Hudson. Musical beauty and information compression: Complex to the ear but simple to the mind? *BMC Research Notes*, 4(1):9+, 2011.

[15] J. Jin, B. Yang, K. Liang, and X. Wang. General image denoising framework based on compressive sensing theory. *Computers & Graphics*, 38(0):382–391, 2014.

[16] R. M. Keller and D. R. Morrison. A grammatical approach to automatic improvisation. In *Proc. SMC '07*, pages 330–337, 2007.

[17] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.

[18] E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *Proc. ACM-SIAM SODA '02*, pages 205–212, 2002.

[19] F. Lerdahl and R. Jackendoff. *A generative theory of tonal music*. The MIT Press, 1983.

[20] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res. (JAIR)*, 7:64–82, 1997.

[21] C.G. Nevill-Manning and I.H. Witten. Online and offline heuristics for inferring hierarchies of repetitions in sequences. In *Proc. IEEE*, volume 88, pages 1745–1755, 2000.

[22] D. Quick and P. Hudak. Grammar-based automated music composition in Haskell. In *Proc. ACM SIGPLAN FARM'13*, pages 59–70, 2013.

[23] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[24] M. Rohrmeier. Towards a generative syntax of tonal harmony. *J. of Math. and Music*, 5(1):35–53, 2011.

[25] J. Schmidhuber. Low-complexity art. *Leonardo*, 30(2):97–103, 1997.

[26] Tim Smith. Fugues of the Well-Tempered Clavier. `http://bach.nau.edu/clavier/nature/fugues/Fugue10.html`, 2013.

[27] M. J. Steedman. A generative grammar for jazz chord sequences. *Music Perception: An Interdisciplinary Journal*, 2(1):52–77, 1984.