Machine learning of symbolic compositional rules with genetic programming: Dissonance treatment in Palestrina

Torsten Anders [1] , **Benjamin Inden** [Corresp. 2]

[1] School of Media Arts and Performance, University of Bedfordshire, Luton, Bedfordshire, United Kingdom

[2] Department of Computer Science and Technology, Nottingham Trent University

Corresponding Author: Benjamin Inden
Email address: benjamin.inden@gmx.de

We describe a method to automatically extract symbolic compositional rules from music corpora that can be combined with each other and manually programmed rules for algorithmic composition, and some preliminary results of applying that method. As machine learning technique we chose genetic programming, because it is capable of learning formula consisting of both logic and numeric relations. Genetic programming was never used for this purpose to our knowledge. We therefore investigate a well understood case in this pilot study: the dissonance treatment in Palestrina's music. We label dissonances with a custom algorithm, automatically cluster melodic fragments with labelled dissonances into different dissonance categories (passing tone, suspension etc.) with the DBSCAN algorithm, and then learn rules describing the dissonance treatment of each category with genetic programming. As positive examples we use dissonances from a given category. As negative examples we us all other dissonances; melodic fragments without dissonances; purely random melodic fragments; and slight random transformations of positive examples. Learnt rules circumstantiate melodic features of the dissonance categories very well, though some resulting best rules allow for minor deviations compared with positive examples (e.g., allowing the dissonance category suspension to occur also on shorter notes).

Machine learning of symbolic compositional rules with genetic programming: Dissonance treatment in Palestrina

Torsten Anders[1] **and Benjamin Inden**[2]

[1]**School of Media Arts and Performance, University of Bedforshire, Luton, United Kingdom**
[2]**Department of Computing & Technology, Nottingham Trent University, Nottingham, United Kingdom**

Corresponding author:
Benjamin Inden[1]

Email address: benjamin.inden@ntu.ac.uk

ABSTRACT

We describe a method to automatically extract symbolic compositional rules from music corpora that can be combined with each other and manually programmed rules for algorithmic composition, and some preliminary results of applying that method. As machine learning technique we chose genetic programming, because it is capable of learning formula consisting of both logic and numeric relations. Genetic programming was never used for this purpose to our knowledge.
We therefore investigate a well understood case in this pilot study: the dissonance treatment in Palestrina's music. We label dissonances with a custom algorithm, automatically cluster melodic fragments with labelled dissonances into different dissonance categories (passing tone, suspension etc.) with the DBSCAN algorithm, and then learn rules describing the dissonance treatment of each category with genetic programming. As positive examples we use dissonances from a given category. As negative examples we us all other dissonances; melodic fragments without dissonances; purely random melodic fragments; and slight random transformations of positive examples.
Learnt rules circumstantiate melodic features of the dissonance categories very well, though some resulting best rules allow for minor deviations compared with positive examples (e.g., allowing the dissonance category suspension to occur also on shorter notes).

INTRODUCTION

Artificial intelligence methods have been used for decades to model music composition (Fernández and Vico, 2013). Two general approaches have attracted particular attention, as they mimic two aspects of how humans learn composition. Firstly, rules have been used for centuries for teaching composition. Algorithmic composition methods model symbolic knowledge with rule-based approaches, formal grammars, and related methods. Secondly, composers learn from examples of existing music. Machine learning (ML) methods to algorithmic composition include Markov chains, and artificial neural networks.

We aim at combining these two approaches by automatically learning compositional rules from music corpora. We use genetic programming (Poli et al., 2008) for that purpose.

The resulting rules are represented symbolically, and can thus be studied by humans (in contrast to, say, artificial neural networks), but the rules can also be integrated into algorithmic composition systems. Extracting rules automatically is useful, e.g., for musicologists to better understand the style of certain corpora, and for composers who use computers as a creative partner (computer-aided composition). For computer scientists, it is a challenging application domain.

The resulting rules can be used in existing rule-based approaches to algorithmic composition where multiple rules can be freely combined, e.g., constraint-based systems (Anders and Miranda, 2011). Rules derived by ML and rules programmed manually can be freely combined in such systems, and rules can address various aspects (e.g., rules on rhythm, melody, harmony, voice leading, and orchestration). Potentially, ML can be used to derive rules from a given corpus of music for aspects where we do not have rules yet, e.g., how to rhythmically and melodically shape the development of a phrase in a certain style.

This paper describes a pilot project within the research programme described above. In this pilot, we automatically extract rules on the treatment of dissonances in Renaissance music using a corpus of movements from Palestrina masses. The treatment of such tones is rather well understood, which helps evaluating results. Nevertheless, this task is far from trivial, as it has to take various musical viewpoints into account (e.g., melodic interval sizes and directions, note durations, and metric positions). Results can be interesting and useful not only for musicologists and composers, but also for the commercially relevant field of music information retrieval to advance the still unsolved problem of automatic harmonic analysis of polyphonic music.

BACKGROUND

Inductive logic programming

Symbolic compositional rules have been extracted by machine learning before, specifically with inductive logic programming (ILP). ILP (Muggleton et al., 2012) combines logic programming with ML in order to learn first-order logic formulas from examples. Background knowledge expressed in logic programs can be taken into account.

ILP has been used for several musical applications. Closely related to our goal is the work of Morales and Morales (Morales and Morales, 1995). Their system learnt standard counterpoint rules on voice leading, namely how to avoid open parallels. Other musical applications of ILP include the learning of harmonic rules that express differences between two music corpora, specifically Beatles songs (pop music) and the Real Book (jazz) (Anglade and Dixon, 2008), and music performance rules for piano (Tobudic and Widmer, 2003) and violin (Ramirez et al., 2010).

Numeric relations are difficult to deduce with ILP, as logic programming in general is very restricted in expressing numeric relations. We are specifically interested in also learning numeric relations besides logic relations, because our experience with constraint-based modelling of music composition makes us aware of their importance for compositional rules. For example, the size of melodic and harmonic intervals is important, and such quantities are best expressed numerically. Besides, we want to use learnt rules later with constraint programming, a programming paradigm with very good support for restricting numeric relations.

Genetic programming

In this project we therefore use another approach. Genetic programming (GP) is a method of ML where a tree structure is learnt by repeated application of random changes (mutation and recombination) and the selection of the best structures among a set of candidates (a population). As such, it is a particular kind of evolutionary algorithm. The candidate tree can be the representation of a computer program or a mathematical equation among other possibilities. Early seminal work on GP has been published by Koza (1992), a more recent practical introduction can be found in Poli et al. (2008).

A particularly important application of GP is symbolic regression. In symbolic regression, a mathematical expression that describes best the given data is inferred. The mathematical expression is unrestricted except that a specified set of building blocks is used – operators like $+$, or standard mathematical functions. The trees that genetic programming builds from these building blocks are representations of such mathematical expressions. Symbolic regression is a powerful method that has been used in various areas of science and engineering (Poli et al., 2008), including a high-profile study where is was used to automatically deduce physical laws from experiments (Schmidt and Lipson, 2009).

GP has been used for music composition before. Spector and Alpern propose a system that automatically generates computer programs for composing four-measure bebop jazz melodies (Spector and Alpern, 1994). The generated programs combine a number of given functions, inspired by Jazz literature, that transform short melodies from Charlie Parker in various ways. The fitness of each program is evaluated by a set of restrictions inspired by Baker (1988), which measure the balance of different aspects (e.g., tonal and rhythmic novelty).

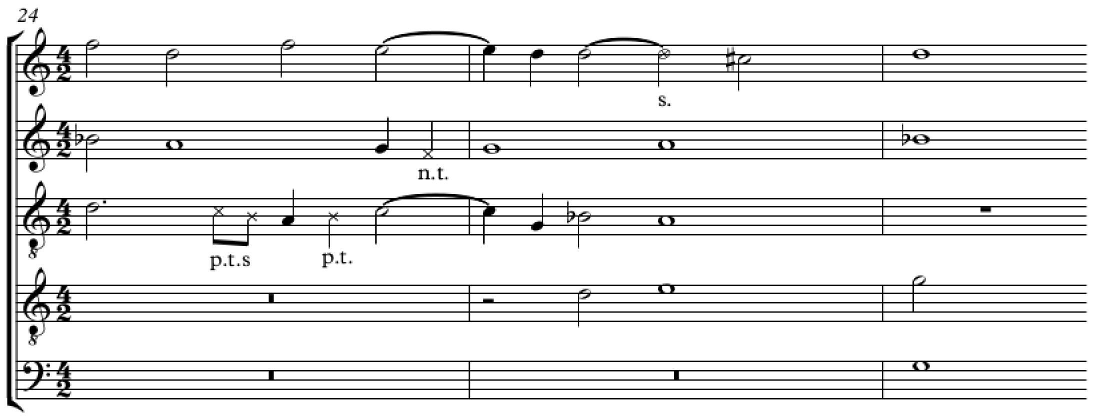

Figure 1. Palestrina excerpt with several dissonances: several passing tones (p.t.), a neighbour tone (n.t.), and a suspension (s.), from Agnus of missa De Beata Marie Virginis (II), measures 24-26

Johanson and Poli (1998) also propose a system that creates computer programs for transforming short melodies, but they allow users to interactively rate the quality of the generated music. This proved a tedious process for users. Therefore they complement the user-rating with automatic fitness raters that learn from the user ratings.

Previous applications of genetic programming for music composition thus aimed at modelling the full composition process, where the fitness function had to judge the quality of the resulting music. Yet, the programs resulting from the evolutionary algorithm are rather short, and they are thus limited in the compositional knowledge they can represent. Previous work therefore composed music by transforming pre-existing music.

Instead, we are interested in learning compositional rules with GP that describe only a certain aspect of the resulting music. Such rules are relevant in their own right as a representation of compositional knowledge that can be complemented with further musical knowledge, e.g., in music constraint programming systems with manually encoded musical rules.

In this situation the fitness function does not need to judge musical quality, but instead only how well the resulting rule fits given positive examples and avoids negative examples.

As far as we know, GP has not yet been used for learning symbolic compositional rules, and therefore in this pilot study we focus on a relatively well understood class of rules.

Dissonances in Palestrina's music

This pilot project studies the dissonance treatment in Palestrina counterpoint with machine learning by automatically generating multiple symbolic rules that each constrain the treatment of a specific dissonance category (passing tones, suspensions etc.).

Jeppesen (1970), the seminal authority on Palestrina counterpoint, distinguishes three roles a dissonance can play in his music. Some dissonances are hardly noticeable on an easy beat used for connecting notes in smooth melodic lines; others occur at an accented beat and are clearly heard; and – more rarely – dissonances can be used for an expressive effect.

As an example, figure 1 shows an excerpt from a Palestrina mass movement[1] with several dissonance categories in close proximity. All dissonances are marked with a crossed notehead, and are labelled with their dissonance category. Passing tones (p.t.) and neighbour tones (n.t.) are short notes on an easy beat that link melodic tones without getting noticed much. By contrast, suspensions (s.) stand out; they occur on a strong beat and typically at longer notes.

The five standard dissonance categories taught in Palestrina counterpoint classes are passing tone, neighbour tone, suspension, anticipation and cambiata. An algorithm for automatically identifying dissonances in Renaissance counterpoint has been proposed (Patrick and Strickler, 1978), but it implements knowledge on the standard dissonance categories and therefore we instead developed a custom algorithm.

[1]The excerpt is from the Agnus of missa De Beata Marie Virginis (II), which is `Agnus_0.krn` in the music21 corpus, and stems from Humdrum.

The actual music of Palestrina contains further dissonance categories, as shown by computational analysis (Sigler et al., 2015), but these are irrelevant for the present study as they either do not occur in the chosen corpus, or they have been rejected by our algorithms.

METHODS

For learning symbolic compositional rules we use a novel methodology that combines multiple established approaches. At first, dissonant notes are automatically labelled in a corpus of music by Palestrina with a custom algorithm. These dissonances are then automatically clustered into different dissonance categories (passing notes, suspensions etc.) with the clustering algorithm DBSCAN (Ester et al., 1996). Finally, a rule is learnt for each of these categories with genetic programming. The rest of this section describes each of these steps in more detail.

Annotation of dissonances

A custom algorithm for dissonance detection in Renaissance music

As a first step we automatically label dissonances in the music using a custom algorithm implemented with the music analysis environment music21 (Cuthbert and Ariza, 2010). For our purposes, the algorithm better leaves a few complex dissonance categories undetected than to wrongly mark notes as dissonances that are actually not. Note that this algorithm does not implement any knowledge of the dissonance categories known to occur in Palestrina's music.

The analysis first "chordifies" the score, i.e., it creates a homorhythmic chord progression where a new chord starts whenever one or more notes start in the score, and each chord contains all the score notes sounding at that time. The algorithm processes those chords and the original score.

The algorithm loops through the chords. If a dissonant chord is found, then it tries to find which note(s) make it dissonant by testing whether the chord becomes consonant if these note(s) are removed.

Dissonances are more likely to occur on short notes in Palestrina, and sometimes multiple dissonant tones occur simultaneously. The algorithm tests individual notes and pairs of simultaneously moving notes whether they are dissonant in an order depending on their duration and a parameter *max_pair_dur*, which specifies the maximum duration of note pairs tested (in our analysis *max_pair_dur* equaled to a half note). In order to minimise mislabelling dissonances, the algorithm first tests all individual notes with a duration up to *max_pair_dur* in increasing order of their duration; then all note pairs in increasing order of their duration; and finally remaining individual notes in order of increasing duration.

Suspensions are treated with special care. If the dissonant note started before the currently tested chord, then that note is split into two notes, which are then tied, and only the second note starting with the current chord is marked as dissonant.

In oder to avoid marking notes wrongly as dissonances, the algorithm does not test the following cases: any note longer than *max_diss_dur*, a given maximum dissonance duration (we set it to a whole tone); and any suspension where the dissonant part of the note would exceed the preceding consonant part, or it would exceed *max_diss_dur*.

For this pilot we arbitrarily selected from the full corpus of Palestrina music that ships with music21 the first 36 Agnus mass movements. All examples in that subcorpus happen to be in $\frac{4}{2}$ meter, but our method does not depend on that.

Evaluation of the dissonance detection algorithm

We evaluated results by examining a sample of scores with marked dissonances. The dissonant detection works rather well, only very few notes are wrongly labelled as a dissonance. Sometimes a suspension is not correctly recognised and instead a wrong note labelled, where the voice proceeds by a skip in shorter notes. Note that such cases are later sorted into an ignored outlier category by the cluster analysis, so that the final clustered data used for the rule learning is near perfect.

Figure 1 shows another example where dissonances are not correctly labelled. The first two passing tones (eighth notes) are correctly labelled in figure 1, but our algorithm would instead label the D in the soprano as a dissonance. The problem here is that when the two correct dissonances are removed, the resulting "chord" A-D is a fourth, which is still considered a dissonance in Renaissance music. Instead, if the soprano D is removed, the remaining chord C-A happens to be consonant.

To be at the save side, the algorithm therefore does not label all dissonances. For example, occasionally more than two dissonances occur simultaneously in Palestrina, e.g., when three parts move with the same short rhythmic values or in a counterpoint of more than four voices. If the algorithm does not find tones that, when removed, leave a consonant chord, then no note is labelled as a dissonance (though the chord is marked for proofreading). Excluding dissonances with the parameter *max_diss_dur* avoided a considerable number of complex cases and otherwise wrongly labelled notes.

Data given to machine learning

Instead of handing the ML algorithm only basic score information (e.g., note pitches and rhythmic values), we provide it with background knowledge (like melodic intervals and accent weights as described below), and that way guide the search process. For flexibility, we considered letting the ML algorithm directly access the music21 score data with a set of given interface functions (methods), but extracting relevant score information in advance is more efficient.

Once dissonances are annotated in the score, only melodic data is needed for the clustering and later the machine learning. For simplicity we only used dissonances surrounded by two consonant notes (i.e., no consecutive dissonances like cambiatas).

In order to control the differences in key between pieces in the corpus we automatically estimate the key of each composition using the Krumhansl-Schmuckler key determination algorithm with simple key correlation weightings (Sapp, 2011). With the key we compute “normalised pitch classes”, where 0 is always the root of the piece, 1 a semitone above the root and so forth.

We determine accent weights using music21’s *getAccentWeight*, where the strongest beat on the first beat of a measure has the weight 1.0; strong beats within a measure (e.g., the third beat in $\frac{4}{2}$ meter) the weight 0.5; the second and fourth beat in $\frac{4}{2}$ meter the weight 0.25 and so on (Ariza and Cuthbert, 2010).

Intervals are measured in semitones, and durations in quarter note lengths of music21, where 1 means a quarter note, 2 a half note and so on.

For simplicity we left ties out in this pilot. Suspensions are simply repeated tones, they are not tied over.

Cluster analysis of dissonance categories

Analysis with DBSCAN algorithm

For each dissonance, we provide the clustering algorithm with the following features: the sum of the durations of the previous, current, and next note (the reason for including this feature instead of including all note durations is explained in the following discussion section); the melodic interval from the previous note, and the interval to the next note; the normalised pitch class of the dissonance; and the accent weight of the dissonance. Before clustering, all data for each feature is normalised such that its mean is 0.0 and its standard deviation is 1.0.

The data is clustered using the DBSCAN algorithm (Ester et al., 1996) as implemented in the scikit-learn library (Pedregosa et al., 2011). This clustering algorithm does not require setting the number of clusters in advance, and can find clusters of arbitrary shape as it is based on the density of points. Here, we set the minimum number of neighbours required to identify a point as a core cluster point to 10, and the maximum neighbour distance to 0.7 based on initial runs and the desire to keep the number of clusters in a reasonable range. Points that lie in low density regions of the sample space are recognised as outliers by DBSCAN, and are ignored in the subsequent rule learning step.

Clustering results and discussion

In order to evaluate the clustering results, we automatically labelled each dissonance in the score with its dissonance category (cluster number), and then created a new score for each cluster number into which all short melodic score snippets that contain this dissonance were collected (one-measure snippets, except where the dissonance occurs at measure boundaries). We then evaluated the clustering results by eyeballing those collections of score snippets.

Initially, the importance of note durations for the clustering was rated too highly, because the clustering took more duration parameters into account (one for every note) than other parameters (e.g., pitch intervals between notes). As a result, one cluster contained primarily dissonances at half notes and another at shorter notes, which was not useful for our purposes. Therefore, we aggregated the duration information, and adjusted the DBSCAN parameters as described above, after which clustering worked very well.

In the selected corpus only the following main dissonance categories are found: passing tones downwards on an easy beat (863 cases); passing tones upwards on an easy beat (643 cases); suspensions on the

Table 1. The features of the automatically derived clusters match traditional dissonance categories

Cluster	Dissonance category	1st interval	2nd interval	metric position	duration
C0	passing tones down	step down	step down	easy beat	up to half note
C1	passing tones up	step up	step up	easy beat	up to half note
C2	suspension on beat 3	repetition	step down	strong beat 3	quarter or half note
C3	suspension on beat 1	repetition	step down	very strong beat 1	quarter or half note
C4	lower neighbour tone	step down	step up	easy beat	up to half note

strong beat 3 in $\frac{4}{2}$ meter (313 cases); suspensions on the strong beat 1 (265 cases); and lower neighbour tones on an easy beat (230 cases).

Table 1 summarises the distinguishing features of the dissonance categories as found in the different clusters, for which we learnt rules. Each row in the table describes a separate dissonance category as recognised by the clustering algorithm. The first interval indicates the melodic interval into the dissonance, and the second the interval from the dissonance to the next note. Metric position and duration are features of the dissonance note itself.

Other dissonance categories like upper neighbour tones, anticipations and cambiatas do not occur in the ML training set. Either they do not exist in the first 36 Agnus mass movements of the music21 Palestrina corpus that we used, or they were excluded in some way. We only use dissonances surrounded by consonances (which excludes cambiatas). Also, we did not use the set of outliers (189 cases), which as expected, has no easily discernible common pattern. Among these outliers are wrongly labelled dissonances, upper neighbour tones, and a few further cases of the above main categories. There are also two small further clusters with lower neighbour tones (25 cases), and passing tones upwards (11 cases) that went through the subsequent rule learning step but were discarded afterwards as they were considered to be too small to be of much interest, and cover categories that are already covered by larger clusters. This simplification of the training set to a small number of dissonance categories was useful for our pilot study.

Learning of rules

Training set

To initiate rule learning, our algorithm compiles a set of three-note-long learning examples with a dissonance as middle note for each identified cluster (dissonance category). All dissonances that have been assigned to that particular cluster are used as positive examples.

Then, four sets of negative examples are generated. Note that the generation of negative examples does not take any knowledge about the problem domain into account. A similar approach can also be used for learning rules on other musical aspects. The first set is a random sample of dissonances that have been assigned to other clusters. The second set is a random sample of three-tone-examples without any dissonance taken from the corpus. The third set consists of examples where all features are set to random values drawn from a uniform distribution over the range of possible values for each feature. The fourth set consists of slightly modified random samples from the set of positive examples. Two variations are generated for each chosen example. Firstly, either the interval between the dissonance tone and the previous note or the interval to the next note is changed by ± 2 (with equal probabilities). Secondly, one of the three note durations is either halved or doubled (with equal probabilities). Both modifications are stored separately in the fourth set of negative examples.

The algorithm aims to create 20% of the number of positive examples for each set of negative examples, but will generate at least 200 examples (100 for the first set due to possible low availability of these examples) and at most 500. These numbers represent mostly a trade-off between accuracy of training/measurement and computation time, and we expect a similar performance if these values are changed within reasonable ranges.

Once all example sets have been constructed, each example receives a weight such that the total weight of the positive examples is 1.0, and the total weight of each of the four sets of negative examples is 0.25 (within each set, the weights are the same for all examples). When measuring classification accuracy of a rule during the learning process, each positive example that is classified correctly counts

+1 times the example weight, whereas each negative example that is erroneously classified as positive example counts -1 times the example weight. Thus, the accuracy score is a number between -1.0 and 1.0, with 0.0 expected for a random classifier.

Please note that a randomly generated negative example can be the same as a positive example with a low probability. Here, we consider this as a small amount of noise in the measurement, but for future experiments it is possible to filter these out at the expense of run time.

Learning process

We use strongly typed genetic programming as implemented in the Python library DEAP[2] (Fortin et al., 2012) with the types float and Boolean (integers are translated to floats). The following functions can occur as parent nodes in the tree representing a learnt rule.

Logical operators: $\vee$ (or), $\wedge$ (and), $\neg$ (not), $\rightarrow$ (implication), $\leftrightarrow$ (equivalence)

Arithmetic operators and relations: $+$, $-$, $\cdot$ (multiplication), $/$ (division), $-$ (unary negation), $=$, $<$, $>$

Conditional: $\mathit{if\_then\_else}(\langle \mathit{boolean} \rangle, \langle \mathit{float} \rangle, \langle \mathit{float} \rangle)$

Terminal nodes in a "rule tree" can be either input variables (like the duration of a note or the interval to its predecessor or successor) or constants. The following input variables can be used in the learnt rules: the duration of the dissonant note ($duration_i$), its predecessor ($duration_{i-1}$) and successor ($duration_{i+1}$); the normalised pitch class of the dissonance; the intervals[3] between the dissonance and its predecessor ($interval_{pre}$) and successor ($interval_{succ}$); and the accent weight of the dissonance ($accentWeight_i$). Additionally, there are the Boolean constants *true* and *false*, as well as ephemeral random constants in the form of integer values between 0 and 13. The notation given here is the notation shown later in learnt rule examples.

There are many choices of operators and parameters that can be used with genetic programming. Here, we follow standard approaches that are commonly used in the GP practitioners' community, and/or are DEAP defaults, unless otherwise noted. The population is created using ramped half-and-half initialisation, after which at each generation the following operators are applied. For selection, we use tournament selection with a tournament size of 3. For mutation, there is a choice between three operators: standard random tree mutation (95% probability), a duplication operator that creates two copies of the tree and connects them using the $\wedge$ operator (2.5%), and a similar duplication operator using the $\vee$ operator (2.5%). For recombination, there is again a choice between standard tree exchange crossover (95%), an operator that returns the first individual unchanged, and a combination of the first and second individual using the $\wedge$ operator (2.5%), and a similar operator using $\vee$ (2.5%). While random tree mutation and tree exchange crossover are commonly used, we designed the other operators to encourage the emergence of rules that are conjunctions or disjunctions of more simple rules, which is a useful format for formal compositional rules. Without these operators, it would be extremely unlikely that a new simple rule could be evolved without disrupting the already evolved one, or that different already evolved rules could be combined as a whole. A static depth limit of 25 is imposed on the evolving trees to avoid stack overflows and exceedingly long execution times.

A population of 100 individuals is evolved for 1000 generations. The fitness assigned to an individual is 10 times its accuracy score (described above) plus 0.001 times the size of its tree. That way, among two rules with equal classification accuracy, the more compact rule has a slight fitness advantage. We introduced this as a measure to slow down the growth of the trees during evolution (known as "bloat control" in the field of genetic programming, although the effect of this particular measure is not very strong). We performed five runs for every cluster. They use the same set of learning examples, but find different rules nonetheless due to the stochastic nature of genetic programming.

After a run is finished, the best rule evolved in that run is output together with its classification accuracy scores.

[2]`https://github.com/deap/deap`

[3]A melodic interval is always computed as the interval between a given note and its predecessor and positive when the next note is higher.

Table 2. Greatest deviation found between features of positive examples (see table 1) and solutions to best rule for each cluster

Cluster	Category	1st interval	2nd interval	metric position	duration
C0	passing tones down	none	none	none	none
C1	passing tones up	none	none	none	none
C2	suspension on beat 3	none	none	none	small[a]
C3	suspension on beat 1	small[b]	none	none	small[a]
C4	lower neighbour tone	small[c]	none	none	none

[a] Can be eighth note. [b] Can be a minor second down. [c] Can be a repetition.

RESULTS

Quantitative evaluation

The quality of the rule learning process as implemented by genetic programming is evaluated by measuring the accuracies of the best evolved rules (see Fig. 2). It can be seen that the accuracies for positive examples are better than 98% in most cases, the accuracies on negative examples from other clusters are mostly better than 99%, the accuracies on negative examples without dissonances are mostly better than 94%, the accuracies on random counterexamples are close to 100%, and the accuracies for modified negative examples are mostly better than 94% but around 89% for the first cluster. When plotting overall accuracy scores against the sizes of the rules' corresponding GP trees (Fig. 3), it can be seen that rules for the same cluster achieve similar accuracy scores despite different sizes. However, across clusters, there seems to be a negative correlation between accuracy and rule size. The most plausible explanation seems to be that larger clusters are more difficult to describe, resulting both in larger rule sizes and lower accuracy scores (Fig. 4).

Qualitative evaluation

We evaluated the suitability of the evolved rules for describing dissonances by using them as melodic constraints in small constraint problems implemented with the music constraint system Cluster Engine,[4] which is a revision of the solver PWMC (Sandred, 2010). Both these solvers are libraries of the visual programming and composition environment PWGL (Laurson et al., 2009). The constraint problems consist of only three notes with the middle note as the dissonance and surrounding rests as padding so that these notes can occur freely on any beat.

For each learnt rule (5 per cluster resulting from the 5 runs reported above) we generated 15 random solutions (an arbitrary number). We examined these solutions in common music notation, and appraised how well they complied with the respective dissonance category. Specifically, we checked whether the metric position and duration of the middle note (the dissonance) and the melodic intervals into and from this note are appropriate.

For each cluster (dissonance category) at least one learnt rule constrains the treatment of the dissonant middle note in a way that either fully complies with the features of the corresponding positive examples (see table 1 again), or is at least very close. In other words, this "best rule" per cluster works either perfectly or at least rather well when used as a constraint for its dissonance category. For the best rule per dissonance category, table 2 reports the greatest deviation found in any solution among a set of 15 random solutions.

Examples of learnt rules

To give a better idea of the kind of rules learnt, figures 5 and 6 show two examples. The rule of figure 5 constrains passing tones upwards and that of figure 6 suspensions on beat 1. These specific rules have been selected, because they are relatively short. Both rules are the best out of their set of 5 in the sense just discussed above, and their results are included in table 2.

The rules generated by DEAP were slightly simplified manually and with Mathematica, and translated into standard mathematical notation for clarity. The names of the input variables, and their possible

[4] `http://sandred.com/Downloads.html`

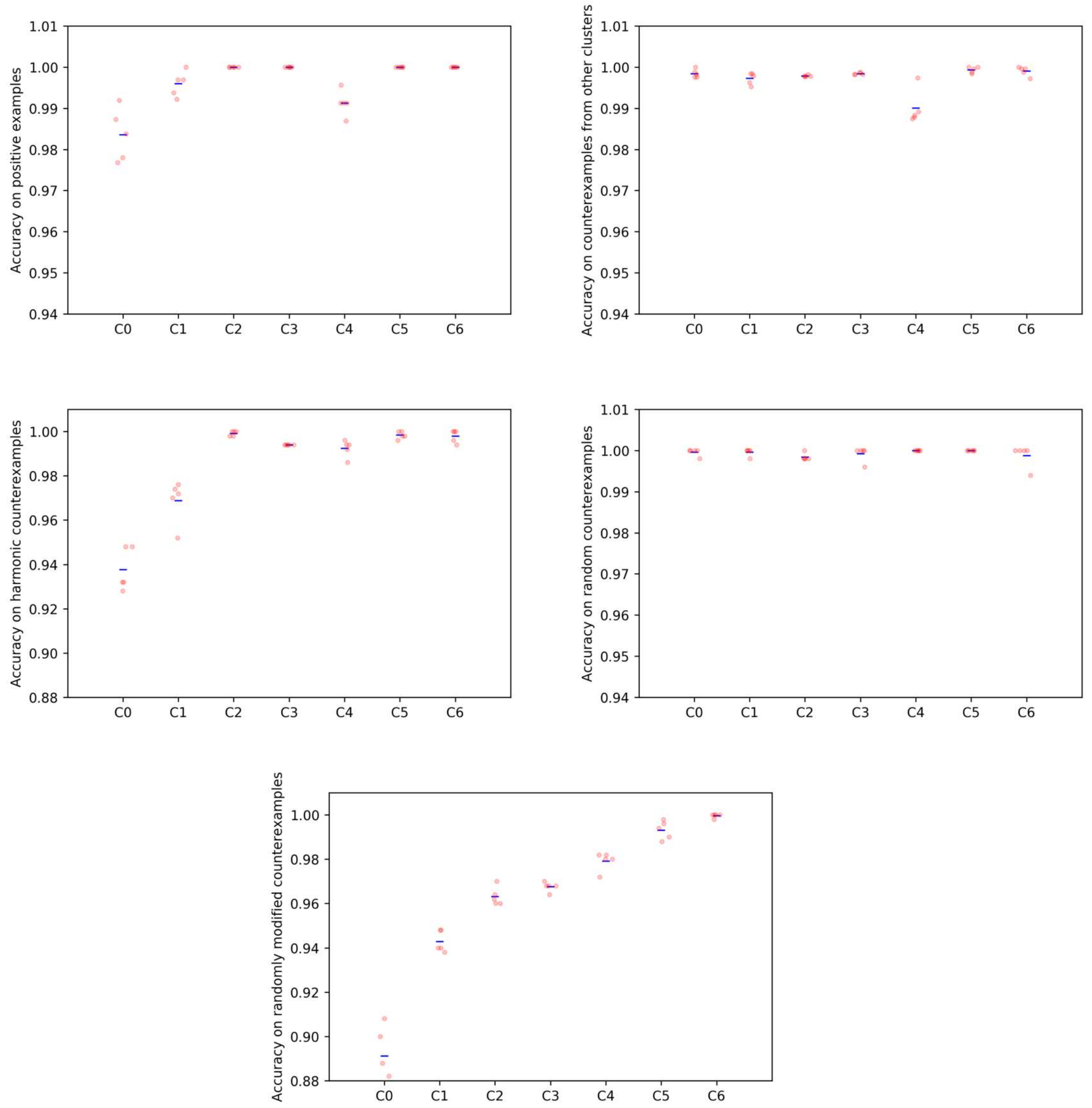

Figure 2. Accuracies achieved by the champions of the genetic programming runs on the various parts of the training sets. The means are indicated by blue bars. C0 - C6 denotes the clusters found by DBSCAN.

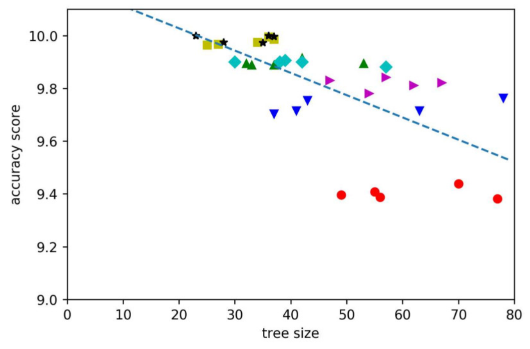

Figure 3. Accuracy score versus tree size for the evolved solutions from all runs. Red dots: cluster 0 (C0); blue lower triangles: C1; green upper triangles: C2; cyan diamonds: C3; magenta right triangles: C4; yellow squares: C5; black stars: C6.

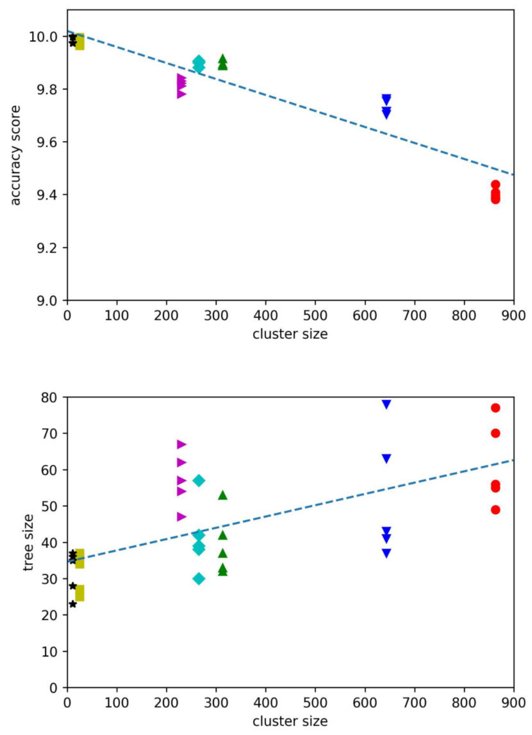

Figure 4. Accuracy and tree size versus cluster size. Clusters are denoted as in Fig. 3.

$$
\begin{aligned}
& duration_{i-1} < duration_i + 3 && (1)\\
\wedge\ & accentWeight_i < 0.5 && (2)\\
\wedge\ & 6 \cdot accentWeight_i < duration_{i-1} && (3)\\
\wedge\ & accentWeight_i < interval_{pre} && (4)\\
\wedge\ & accentWeight_i < interval_{succ} && (5)\\
\wedge\ & duration_i < 3 && (6)\\
\wedge\ & accentWeight_i \cdot duration_{i+1} < duration_i && (7)\\
\wedge\ & interval_{pre} < 3 && (8)\\
\wedge\ & interval_{succ} < 3 && (9)
\end{aligned}
$$

Figure 5. Learnt rule example: passing tones upwards

$$
\begin{aligned}
& 2 \geq |interval_{succ}|\\
\wedge\ & accentWeight_i \geq 1\\
\wedge\ & 2 < duration_i \vee duration_{i-1} \geq 2\\
\wedge\ & 2 \geq duration_i \vee duration_{i-1} < 2\\
\wedge\ & interval_{pre} < accentWeight_i\\
\wedge\ & interval_{pre} > interval_{succ}
\end{aligned}
$$

Figure 6. Learnt rule example: suspension on beat 1

values, have been explained above, but we will briefly revise them below for the reader's convenience.

As an example, let us analyse the first rule, which constraints upwards passing tones (figure 5). Remember that for this dissonance category both intervals lead upwards stepwise, the dissonance occurs on an easy beat, and its duration is up to a half note (table 1). This rule constrains all those aspects exactly (table 2).

The rule enforces that both the melodic interval into the dissonance and out of it, $interval_{pre}$ and $interval_{succ}$, are positive (upwards): they are both greater than $accentWeight_i$, see equations (4) and (5), and $accentWeight_i$ is always greater than 0 by its definition. Intervals are integers measured in semitones. Both intervals are less than 3, see equations (8) and (9). So, in summary the intervals are positive (upwards), but at most 2 semitones (steps).

The rule constrains dissonances to an easy beat. For the first beat of a measure $accentWeight_i$ is 1.0, for the third beat in $\frac{4}{2}$ it is 0.5, of the second and forth beat it is 0.25 and so on. The rule constrains the accent weight of the dissonance to less than 0.5, i.e., an easy beat.

The duration must be a half note or shorter. Durations are measured in music21's quarterlengths, where 1 represents a quarter note. The duration of the dissonance must be less than 3, which corresponds to a dotted half note (6), hence it can be a half note at most.

Other expressions in this rule happen to be less meaningful, and could be considered bloat.

The other rule in Figure 6 can be analysed similarly. We leave that to the reader.

DISCUSSION

In the present paper we describe a method based on genetic programming that extracts symbolic compositional rules from a music corpus so that resulting rules can be used in rule-based algorithmic composition systems. In this pilot study we extracted rules that detail the dissonance treatment in compositions by Palestrina. We derived rules for the following five dissonance categories (automatically derived clusters): passing tones on an easy beat upwards and downwards; lower neighbour tones on an easy beat; and suspensions on the strong beat one and beat three in $\frac{4}{2}$ meter. Learnt rules are typically able to recognize between 98% and 99% of the positive training examples, while excluding between 89% and 100% of the counterexamples depending on counterexample category and cluster, with better results for smaller clusters.

Multiple rules learnt for the same cluster differ in their accuracy when used as a constraint for music generation: the music generated with these rules can be more or less close to the features of the positive examples (see table 1). Table 2 only reports the accuracy of the best rule per cluster. Some other rules for the same cluster are much less accurate, but nevertheless obtain a very similar overall weighted score in the learning process. Currently, we lack an algorithm for measuring the accuracy of a rule in terms of how similar generated music restricted by that rule is to its positive examples. Such an algorithm would be very useful to contribute to the fitness calculation of rules during the learning process.

The accuracy of resulting rules can also be improved by taking further background knowledge into account. For example, some resulting rules allow for syncopations in dissonance categories where these would not occur in Palestrina, e.g., at a passing tone. Providing the rule learning algorithm with an extra Boolean feature whether the dissonant note is a syncope or not will likely avoid that.

The negative examples in the training set for the rule learning have a great impact on the accuracy of resulting rules. For example, the inclusion of slightly modified transformations of positive examples clearly improved the accuracy as compared to preliminary experiments. A closer investigation into the impact of automatically generated negative examples on the accuracy of resulting rules could lead to further improvement. For example, so far we only used slight random variations of the note durations and melodic intervals to generate negative examples, but variations of further parameters could also be useful (e.g., negative examples with shifted metric positions could also restrict syncopations).

A further improvement could probably be obtained by post-processing large clusters generated by DBSCAN with another clustering algorithm that is not based on density alone, or by DBSCAN with a smaller setting for maximum neighbour distance, to split them up into smaller clusters, for which learning a suitable rule should be easier.

Acknowledgments

We thank Dmitri Tymoczko, who suggested the initial idea for an algorithm that detects dissonances in Renaissance music.

REFERENCES

Anders, T. and Miranda, E. R. (2011). Constraint Programming Systems for Modeling Music Theories and Composition. *ACM Computing Surveys*, 43(4):30:1–30:38. Article 30.

Anglade, A. and Dixon, S. (2008). Characterisation of Harmony With Inductive Logic Programming. In *Proceedings of the Ninth International Society for Music Information Retrieval Conference, ISMIR*, pages 63–68.

Ariza, C. and Cuthbert, M. S. (2010). Modeling Beats, Accents, Beams, and Time Signatures Hierarchically with music21 Meter Objects. In *Proceedings of the International Computer Music Conference*, pages 216–223.

Baker, D. (1988). *David Baker's Jazz Improvisation*. Alfred Publishing, revised edition.

Cuthbert, M. S. and Ariza, C. (2010). music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data. In *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010*, pages 637–642, Utrecht, Netherlands.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press.

Fernández, J. D. and Vico, F. (2013). AI Methods in Algorithmic Composition: A Comprehensive Survey. *Journal of Artificial Intelligence Research*, 48:513–582.

Fortin, F.-A., Rainville, D., Gardner, M.-A. G., Parizeau, M., Gagné, C., and others (2012). DEAP: Evolutionary Algorithms Made Easy. *The Journal of Machine Learning Research*, 13(1):2171–2175.

Jeppesen, K. (1970). *The Style of Palestrina and the Dissonance*. Dover Publications. Reprint of a 1946 publication by Oxford University Press.

Johanson, B. and Poli, R. (1998). Gp-music: An interactive genetic programming system for music generation with automated fitness raters. In *Proceedings of the Third Annual Conference*, pages 181–186. MIT Press.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Laurson, M., Kuuskankare, M., and Norilo, V. (2009). An Overview of PWGL, a Visual Programming Environment for Music. *Computer Music Journal*, 33(1):19–31.

Morales, E. and Morales, R. (1995). Learning Musical Rules. In Widmer, G., editor, *Proceedings of the IJCAI-95 International Workshop on Artificial Intelligence and Music, 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada.

Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., and Srinivasan, A. (2012). ILP turns 20. *Machine Learning*, 86(1):3–23.

Patrick, P. H. and Strickler, K. (1978). A Computer-Assisted Study of Dissonance in the Masses of Josquin Desprez. *Computers and the Humanities*, 12(4):341–364.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay (2011). Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu.com.

Ramirez, R., Perez, A., Kersten, S., Rizo, D., Roman, P., and Inesta, J. M. (2010). Modeling violin performances using inductive logic programming. *Intelligent Data Analysis*, 14(5):573–585.

Sandred, Ö. (2010). PWMC, a Constraint-Solving System for Generating Music Scores. *Computer Music Journal*, 34(2):8–24.

Sapp, C. S. (2011). *Computational Methods for the Analysis of Musical Structure*. PhD thesis, Stanford University.

Schmidt, M. and Lipson, H. (2009). Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85.

Sigler, A., Wild, J., and Handelman, E. (2015). Schematizing the Treatment of Dissonance in 16th-Century Counterpoint. In *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015*, pages 645–651, Málaga, Spain.

Spector, L. and Alpern, A. (1994). Criticism, culture, and the automatic generation of artworks. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, pages 3–8.

Tobudic, A. and Widmer, G. (2003). Relational IBL in music with a new structural similarity measure. In *Proceedings of the International Conference on Inductive Logic Programming*, pages 365–382. Springer.